DOKUMENTATION
LABOR- & FORSCHUNGSPROJEKT
EvaGrip2

Evaluation von 6 DoF Eingabegeräten

Im Studiengang

Mediensysteme

SS2004 – Oktober 2004

Robert Gerling

Manuel Gleisberg

André Kunert

Verena Skuk

Carsten Tetens

Betreuer

Jun. Prof. Dr. Anke Huckauf

Prof. Dr. Bernd Fröhlich

Dipl.-Inf. (FH) Jan Hochstrate

Abstract

Diese Dokumentation beschreibt in ausgearbeiteter Form die Ergebnisse und Tätigkeiten des Labor- & Forschungsprojekts EvaGrip2. EvaGrip2 ist das Nachfolgeprojekt von EvaGrip, welches sich allerdings mit der Evaluation von 12 DoF Eingabegeräten beschäftigte. Der Name setzt sich aus „Eva“ für Evaluation und dem englischen „grip“ (greifen) zusammen. Grip ist ein Hinweis auf das Thema des Forschungsprojekts, die Evaluation von neuen 6 DoF Eingabegeräten.

Es werden zwei neue Gerätekonzepte vorgestellt, welche isotonische Rotation mit isometrischer Translation verbinden. Der Kugelfisch besteht aus einer gewöhnlichen 3 DoF Trackball Kugel, welche in einem elastischen Rahmen aufgehängt ist. Die elastische Aufhängung erlaubt feine Translationen des Trackballs in alle drei räumliche Direktionen.

Die Kugelmaus arbeitet ähnlich, mit dem Unterschied, dass die Kugel auf einem beweglichen Sockel sitzt. Es ist erforderlich, das der Nutzer zwischen Rotation der Kugel und der Translation des Sockels umgreift.

Die Geräte werden alle mit den Fingerspitzen manipuliert und erlauben so eine feinfühlige Interaktion mit den virtuellen Objekten. Eine andere wichtige Eigenschaft der Geräte ist die effektive Trennung von isotonischer Rotation und isometrischer Translation.

Eine Benutzerstudie offenbart, das die neuen Geräte signifikant besser Leistungen in einem DockingTask bringen als die handelsübliche SpaceMouse™. Subjektive Bewertungen bestätigen diese Ergebnisse.

Keywords

Evaluation von Eingabegeräten, 6 DOF, Eingabegeräte, Input Devices, Docking, isotonische Eingabegeräte, isometrische Eingabegeräte, Positionssteuerung, Geschwindigkeitssteuerung, Simultanität.

Besonderer Dank

An dieser Stelle möchten wir unseren Betreuern Jun. Prof. Dr. Anke Huckauf, Prof. Dr. Bernd Fröhlich und Dipl.-Inf. (FH) Jan Hochstrate für ihre durchgehende Unterstützung während des Projekts danken.

Inhaltsverzeichnis

1. Einführung

1.1 EvaGrip2

EvaGrip2 beschäftigte sich weiterführend mit der Evaluierung von Eingabegeräten, im Gegensatz zu EvaGrip, jetzt speziell mit 6 DoF („Six Degrees of Freedom“), also Geräten mit 6 Freiheitsgraden. Dies ermöglicht dem Benutzer die freie Interaktion im 3D-Raum, d.h. die Translation in x-, y-, z-Richtung, sowie die Rotation um diese drei Achsen.

Die Herausforderung für die Weiterführung von EvaGrip bestand darin, alternative 6-DoF Gerätekonzepte zu kommerziell verfügbaren Geräten zu finden. Dabei ging es speziell um den Vergleich von drei Prototypen (kleiner & großer Kugelfisch, Kugelmaus), die an der Bauhaus Universität Weimar (BUW) entwickelt worden sind, mit der kommerziell verfügbaren SpaceMouse.

Im Rahmen unseres Projektes ergaben sich zwei Hauptaufgaben. Zum einen die Entwicklung einer geeigneten Testaufgaben unter zu Hilfenahme der Entwicklungs-umgebung Avango. Zum Anderen der Test von Versuchspersonen und die damit verbundene Auswertung der Ergebnisse mit Hilfe von SPSS, um so einen Rück-schluss auf die Fähigkeiten der Geräte zu erlangen.

Des Weiteren musste ein Messverfahren entwickelt werden, mit der die 3 Rotations–DoFs mathematisch korrekt von einer Kugel erfasst. Bisherige Messtechniken (TrackballTM) konnten nur 2 DoFs von einer Kugel erfassen.

1.2 Begriffe und Grundlagen

Eingabegeräte kann man zum einen nach dem Gerätetyp und zum anderen nach dem Steuerungsmodus einteilen.

1.2.1 Gerätetypen

Gerätetypen lassen sich in vier Kategorien einteilen. In isotonische, isometrische, und elastische Geräte.

Isotonische (Synonyme: anisometric, displacement device, unload device, free moving device) Geräte sind Geräte ohne spürbaren bzw. mit konstanten Widerstand zum Beispiel die Desktop-Maus (Abb.1.1).

Abb. 1.1: Desktop-Maus.

Von *isometrisch* (Synonyme: pressure device, force device) spricht man bei Geräten, die auf Krafteinwirkung reagieren, jedoch keine wahrnehmbare Auslenkung spüren lassen. Das kann zum Beispiel ein Spaceball sein (Abb. 1.2).

Abb. 1.2: Spaceball.

Elastische oder auch *federnde* Eingabegeräte besitzen einen variierenden Widerstand, d.h. Erhöhung des Gerätewiderstandes mit der Auslenkung, wie es beim Joystick (Abb. 1.3) oder der Spacemouse (Abb. 1.6) der Fall ist.

Abb. 1.3: Joystick.

1.2.2 Steuerungsmodi

Die Steuerungsmodi kann man in die Position Control (Positionssteuerung) und die Rate Control (Geschwindigkeitssteuerung) unterteilen.

Bei der Positionssteuerung manipuliert der Benutzer die Objektposition direkt. Sie besitzt eine konstante 1 zu 1 bzw. 1 zu k Übersetzung vom Input zum Output, man nennt dies eine Transferfunktion 0. Ordnung. (Abb. 1.4) Vorteil dieses Modus ist die intuitive Steuerung und Kontrolle der Geräte. Nachteil ist jedoch, das alle Bewegungen,

Abb. 1.4: Beispiel einer Transferfunktion 0. Ordnung

auch die unfreiwilligen, übertragen werden. Außerdem ist eine Geschwindigkeitskontrolle nur schwer möglich, da es häufig zu sogenannten „jerky motions“ (ruckartigen Bewegungen) kommen. Ein weiterer Nachteil ist der limitierte Eingabebereich bzw. Eingaberadius, so dass bei einigen Geräten ein „reclutching“ (Auskoppeln) notwendig wird. Auskoppeln bedeutet, dass man zum Beispiel die Maus vom Mauspad anheben und in eine neue unlimitierende Position bringen muss, um dann die Bewegungssteuerung wieder aufnehmen zu können.

Bei der Geschwindigkeitssteuerung werden die Benutzereingaben direkt auf die Objektgeschwindigkeit abgebildet. Dies bezeichnet man als Transferfunktionen 1. Ordnung (Abb. 1.5).

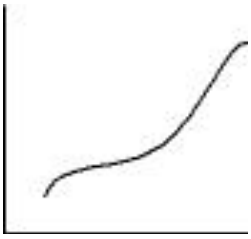

Abb. 1.5: Beispiel einer Transferfunktion 1. Ordnung

Diese haben den Vorteil, das man einen Low Pass Filter Effekt erzeugen kann, d.h. es ist so möglich unfreiwillige Eingaben unterhalb eines gesetzten Schwellenwerts herauszufiltern.

Dadurch das der Benutzer die Objektgeschwindigkeit direkt beeinflussen kann, sind glattere Bewegungen möglich. Ein weiterer Vorteil dieser Eingabemethode ist der nicht limitierte Eingabebereich, welcher ein Auskoppeln überflüssig macht. Als Beispielgerät wäre hier die Spacemouse zu nennen (Abb. 1.6).

Abb. 1.6: SpaceMouse

1.2.3 Zhai's DockingTask

Auf dem Gebiet der mehrdimensionalen Eingabegeräte ist Shumin Zhai neben Maurice R. Masliah eine wichtige Referenz für die Evaluation, wie EvaGrip sie betreibt. Im Fokus von EvaGrip2 stand das erste Experiment der Dissertation von Shumin Zhai und diente als Grundlage für unser Vorhaben. Im Folgenden wird das Experiment, wie Shumin Zhai es durchgeführt hatte, vorgestellt und später der modifizierte Nachbau, wie er in unserer Studie zum Einsatz kam.

Voran steht in diesem Experiment von Shumin Zhai die Fragestellung:
Welche Effekte haben Gerätewiderstand und Transferfunktion auf das Manipulationsverhalten von Versuchspersonen in 6 DoF Umgebungen?

Untersucht wurden dabei in vier Kombinationen die folgenden Parameter:

- Gerätewiderstand: isometrisch, isotonisch
- Transferfunktion: Geschwindigkeitssteuerung, Positionssteuerung

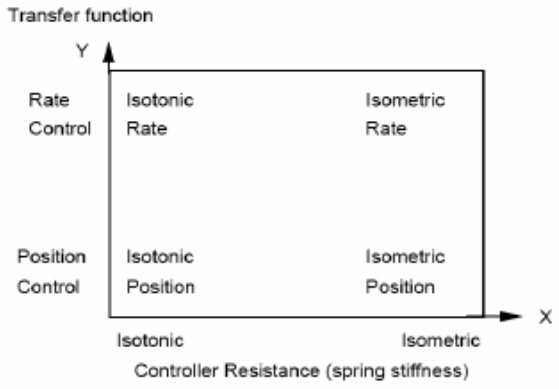

Abb. 1.7: Der Versuchsraum des Experiments.

Ziel von Zhai's DockingTask ist es, so schnell wie möglich einen 3D Cursor mit einem 3D Target in Übereinstimmung zu bringen. Cursor und Target sind Tetraeder von gleicher Größe. Beide Tetraeder werden als WireFrame Modell dargestellt, d.h. die Strukturen im hinteren Teil des Tetraeders werden nicht von den vorderen Flächen verdeckt. Der gesamte Tetraeder ist immer sichtbar (Abb. 1.8).

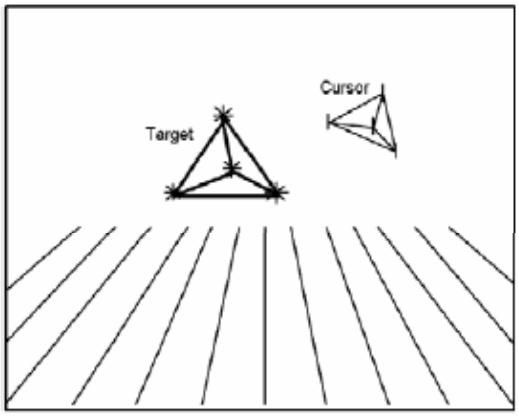

Abb. 1.8: Docking Task von S. Zhai mit 2 Wireframe Tetraedern.

Die Kanten der Tetraeder sind mit verschiedenen Farben versehen, um sicherzustellen, dass es nur eine einzige Übereinstimmung in der Orientierung der Tetraeder zueinander gibt. Der Target-Tetraeder wird an den Ecken mit Sternen, der Cursor-Tetraeder mit Linien dargestellt. Diese unterschiedlichen Markierungen sollen zum einen die Aufgabe erfüllen, Target und Cursor optisch unterscheiden zu können, zum anderen repräsentiert die Sternmarkierung am Target-Tetraeder zugleich den Toleranzspielraum jedes Eckpunktes. D.h. die Cursor Eckpunkte müssen nicht exakt auf den Target Eckpunkten liegen. Es reicht, wenn sie sich in dem vom Stern markierten Bereich um den jeweiligen Eckpunkt befinden.

Der Target-Tetraeder verbleibt während des gesamten Experiments im Zentrum des Bildschirms. Der Cursor-Tetraeder erscheint indessen zu Beginn jedes Trials zufällig an einem von vier fest definierten Startpunkten in der Peripherie des Bildschirms. Wenn im Laufe des Trials ein Eckpunkt des Cursors, in den vom zugehörigen Target Eckpunkt umgebenen Sternbereich eintritt, verändert dieser Stern seine Farbe. Somit wird angezeigt, dass dieser Eckpunkt richtig positioniert wurde. Sind alle Eckpunkte für einen Zeitraum von mindestens 0.8 Sekunden richtig positioniert, gilt die Docking Aufgabe als erfüllt.

1.2.3.1 In Zhai's Test verwendete 6 DoF Eingabegeräte

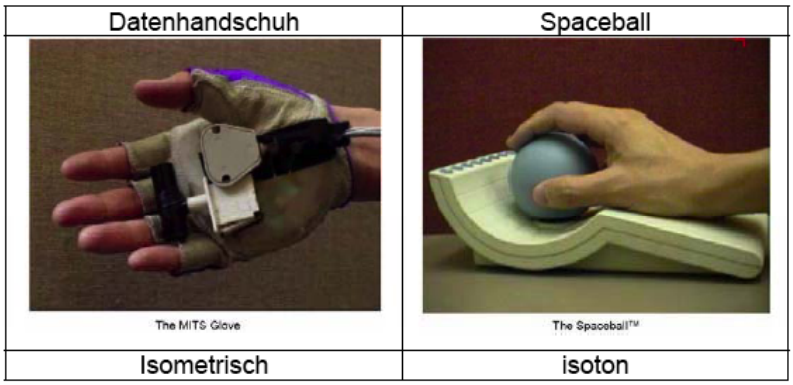

Abb.1.9: Datenhandschuh & Spaceball.

1.2.3.2 Zhai's Versuchsdesign & Versuchsaufbau

Der Test wurde in einem stereoskopischen Setup durchgeführt. Die Testperson sitzt dabei ca. 60 cm vom Bildschirm entfernt (Abb. 1.10).

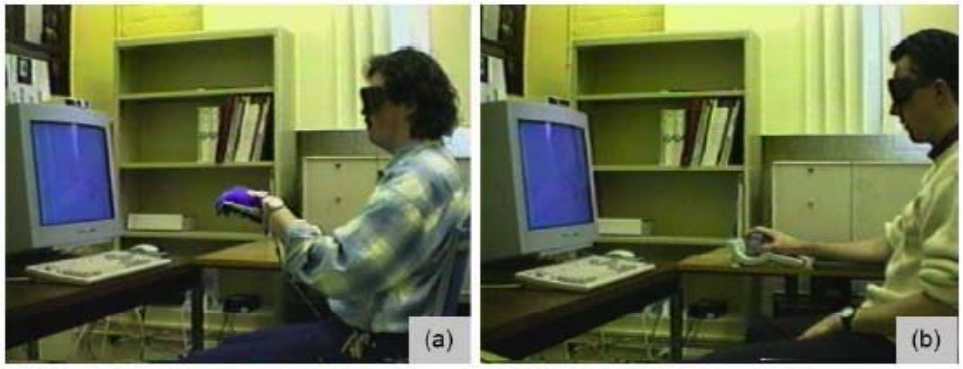

Abb. 1.10: Testaufbau.

Die Aufgaben wurden von 8 Testpersonen im Within-Subject-Design bearbeitet, d.h. jede Testperson testet alle Ausprägungen der unabhängigen Variablen (jedes Gerät, jeden Steuerungsmodus). Es wurden drei präventive Maßnahmen vorgenommen, um Ermüdung und "skill transfer" zu vermeiden. Die vier verschiedenen Manipulationstechniken werden an vier verschiedenen Tagen getestet mit nur 10 Minuten Training zwischen den 12 Wertungstrials (Abb. 1.11). Die Reihenfolge der 4 Steuerungsmodi ist ausgeglichen verteilt über die 8 Versuchspersonen.

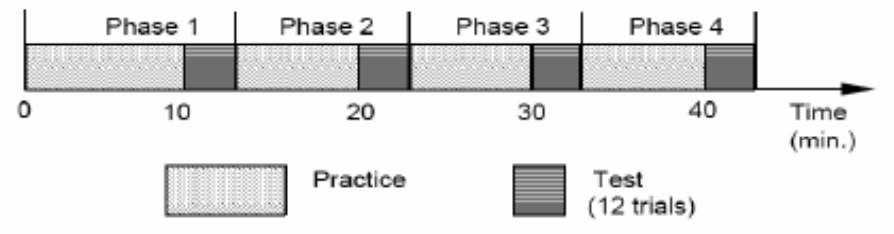

Abb. 1.11: Einteilung der Übungs- und Testphasen.

1.2.3.3 Resultate und Schlussfolgerungen der Tests

Zwischen Isometrischer Geschwindigkeitssteuerung und isotonischer Positionssteuerung gibt es keinen signifikanten Performanceunterschied, beide Steuerungsmodi erreichen die kürzesten TCTs. Isometrische Positionssteuerung liefert mit Abstand die schlechtesten Zeiten.

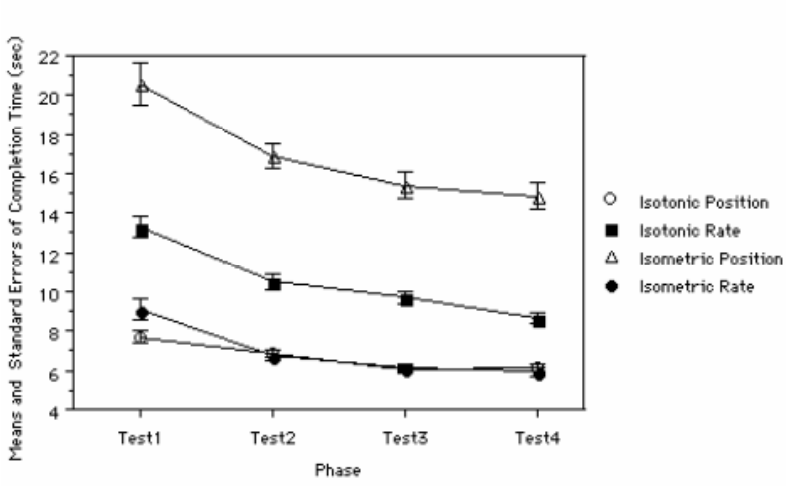

Abb. 1.12: Mittelwerte TCT der isometrischen Geschwindigkeitssteuerung und der isotonischen Positionssteuerung.

Mit isometrischer Geschwindigkeitskontrolle sind die besten Einzelzeiten erreichbar, aber etwas größere Streuung der TCTs als bei der isotonischer Positionssteuerung. Isotonische Geschwindigkeitssteuerung zeigt vereinzelt schnelle Zeiten, hat aber durch einige langsamere Versuchspersonen eine insgesamt schlechtere Performance als isotonische Positionssteuerung und isometrischer Geschwindigkeitskontrolle. Isometrische Positionssteuerung zeigt durch hohe TCT und sehr hohe Streuung einiger Versuchspersonen die schlechteste Performance aller vier möglichen Modi. Die Ergebnisse wurden auch durch die subjektive Beurteilung der Versuchspersonen gestützt.

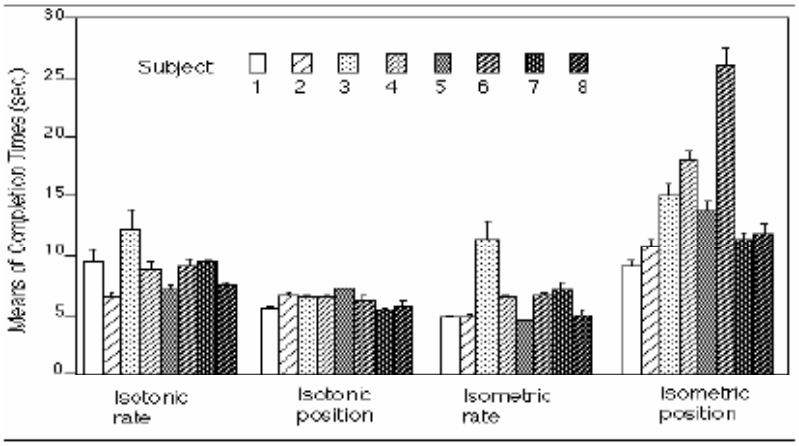

Abb. 1.13: Personenwerte TCT der isometrischen Geschwindigkeitssteuerung und der isotonischen Positionssteuerung

Zusammenfassend lassen die Ergebnisse die Schlussfolgerungen zu, dass bei der Positionssteuerung isotonische Geräte besser als isometrische sind und es bei der Geschwindigkeitssteuerung genau umgekehrt ist. Zudem ist isotonische Positionskontrolle intuitiver als isometrische Geschwindigkeitssteuerung und damit schneller zu erlernen.

2. Die Eingabegeräte

2.1 Referenzgerät: SpaceMouse

Eingabegeräte, welche 6 DoF zur Manipulation im 3D zur Verfügungen stellen werden größtenteils in Modellierungs- und Konstruktionsprogrammen verwendet (z.B. 3Dsmax, Maya, CAD-Anwendungen) um dort die Sicht (view) auf eine Szene einstellen zu können. Typischerweise werden diese Eingabegeräte mit der rezessiven Hand bedient, während die dominante Hand mit Maus oder Tastatur etwas in der Szene manipuliert.

Wenn man den momentanen Markt der 6 DoF Eingabegeräte betrachtet kommt man an der Firma 3DConnexions (http://www.3dconnexions.com) und deren Eingabegeräten nicht vorbei. Diese Tochterfirma von Logitech (http://www.logitech.com) hat mit ihren Geräten SpaceMouse (modern und klassik) (Abb. 2.1), Spaceball (Abb. 2.2), SpaceMouse Traveller (Abb. 2.3) zurzeit eine marktbeherrschende Stellung in diesem Segment inne.

Abb. 2.1: SpaceMouse.

Abb. 2.2: SpaceBall.

Abb. 2.3 SpaceTraveler.

Ihren Ursprung fand die Technik in der Raumfahrt, da sie auf Wunsch der NASA Anfang der 90er Jahre entwickelt wurde, um damit den Roboterarm des Spaceshuttles zu steuern. Die dort entwickelte Technik wurde in Form der SpaceMouse auf den Benutzermarkt gebracht. Der SpaceBall und der SpaceTraveller sind als Fortführungen des SpaceMouse-Konzepts entstanden. Die drei Geräte sind in die Klasse der isometrischen Eingabegeräte (Kapitel 1.2.1) einzuordnen, genauer gesagt handelt es sich sogar um elastische Eingabegeräte. Typischerweise werden solche Eingabegeräten in Verbindung mit einer Geschwindigkeitssteuerung (Kapitel 1.2.2) eingesetzt.

Uns interessierte nun im Speziellen die SpaceMouse, da sie das Referenzeingabegerät in diesem Bereich darstellt, welches jedoch wie wir finden ein paar Nachteile aufweißt.

Es wird nun der prinzipielle Aufbau und die Funktionsweise der SpaceMouse dargestellt, und anschließend auf die daraus resultierenden Nachteile eingegangen.

Die SpaceMouse besteht aus einem Sockel auf dem ein Puck aufgesetzt ist. Unter diesem Sockel befindet sich die SpaceMouse-Sensorik. Außerdem befinden sich noch einige Knöpfe, zum Umschalten verschiedenster Steuermodi auf dem Sockel. Der Puck der SpaceMouse kann nun ausgelenkt, gekippt bzw. verdreht werden. Diese Manipulation am Gerät wird durch die Sensorik aufgenommen und üblicherweise über die serielle Schnittstelle (RS 232) an den Rechner weitergeben. Ein Auslenken des Pucks bewirkt nun eine Translation entlang der entsprechenden Achse, wohingegen ein Kippen bzw. Verdrehen eine Rotation um die entsprechende Achse hervorruft (Abb. 2.4). Natürlich sind auch kombinierte Rotationen bzw. Translationen möglich, d.h. die Rotations- bzw. Translationsachse kann frei im Raum liegen und muss nicht mit einer der Achsen des Gerätekoordinatensystems zusammenfallen. Somit stellt die SpaceMouse alle sechs DoF gleichzeitig zur Verfügung die mit einer Hand bedient werden können.

Abb. 2.4: Steuerungsprinzip der SpaceMouse (hier klassik).

2.2. Motivation für neue Geräte

Wenn das Bedienkonzept verstanden ist wird klar, dass man auch simultan translieren und rotieren kann. Dies scheint im ersten Moment ein Vorteil zu sein, jedoch ist dieser Effekt nicht immer erwünscht. Es kann so, je nach Sensibilität der Sensorik und des jeweiligen Steuermodi, zu ungewollten Rotationen bei einer beabsichtigten Translation und umgekehrt kommen. Grade für Personen die keine Erfahrung mit der SpaceMouse haben ist dies am Anfang irritierend und

demotivierend. Es gibt jedoch die Möglichkeit Modi, bei denen nur die Rotations bzw. nur die Translationsinformationen der SpaceMouse verarbeitet werden, zu wählen. Doch da wir mit allen 6 Dof arbeiten wollen, ist die Tatsache der ungewollten Überlagerung von Manipulationen unser erster Kritikpunkt an der SpaceMouse.

Geübte Benutzer können allerdings auch Vorteile aus der möglichen Überlagerung der beiden Manipulationsarten (Rotation und Translation) ziehen.

Ein weiterer Nachteil der SpaceMouse, und somit unser zweiter Kritikpunkt ist die unnatürliche und wenig intuitive Bedienung, um mit der SpaceMouse Rotationen zu produzieren. Hierfür ist es notwendig den Puck um die gewünschte Rotationsachse zu verdrehen bzw. zu kippen. Dies ruft grade bei ungeübten Benutzern Irritationen hervor. Der Zusammenhang zwischen verdrehen bzw. kippen des Pucks und der Rotation des virtuellen Objektes ist nicht offensichtlich. Das „Treffen“ der gewünschten Rotationssachse ist schwierig und mit einiger Überlegung und vor allem Übung verbunden.

Genau dies ist der Ansatzpunkt der Überlegungen, die zu den drei Eingabegeräten führten die an der BUW entstanden sind. Der Grundgedanke der hinter allen drei Geräten steckt, ist die Steuerung der Rotation intuitiver und natürlich zu gestalten. Nahe liegend wurde dies durch die Manipulation einer Kugel, welche mit den Fingern frei rotiert werden kann, realisiert. Die Rotationen der Kugel wird durch Sensoren abgenommen und dann entsprechenden 1:1 oder um einen Faktor k verstärkt auf das virtuelle Objekt übertragen. Dies stellt eine sehr natürliche und intuitive Art der Bedingungen dar, da die Manipulation am realen Objekt „Kugel“ direkt auf das virtuelle Objekt übertragen wird. Durch die Augen und die Finger hat der Benutzer ein propriozeptives Feedback über seine Interaktion mit der Kugel. So bekommt er eine Vorstellung, wie sich seine Aktionen auf das virtuelle Objekt auswirken und genau diese Vorstellung wird dank des direkten 1:1 bzw. 1:k Mappings umgesetzt.

Im Folgenden sollen nun die drei an der BUW entstandenen Geräte, ihre Technik, ihre Funktionsweise, ihre Stärken und Schwächen vorgestellt werden.

2.3 Großer Kugelfisch

Die Reihenfolge in der die Geräte vorgestellt werden, ist nicht zufällig, sondern spiegelt die Entstehungsreihenfolge wieder. Zuerst wurde der große Kugelfisch gebaut (Abb. 2.5). Schon in der ersten Version (das Gerät wurde einige Male überarbeit und verbessert, unter anderem durch andere Sensorik) ist der grundlegende Aufbau des Kugelfisches erkennbar. Am Gehäuse ist eine Fassung befestigt. Die Fassung besitzt einen halbkreisförmigen Ausschnitt, in welchem die Kugel (55 mm Durchmesser) eingelassen ist (Abb. 2.6, Abb. 2.7). Die Fassung ist mit Potentiometern verbunden, welche sich im Gehäuseinnern verbergen. Es ist ein gewisser Spielraum vorhanden so dass sich die Fassung in alle drei Richtungen (x, y, z) verschieben lässt. Diese Verschiebung bewirkt eine Auslenkung der Potentiometer, mittels der man die Verschiebung erfassen kann um sie entsprechend als Translation auf das virtuelle Objekt zu mappen. Auch hier lässt sich durch das richtige (logische) Mapping eine natürliche und intuitive Benutzungssituation erzeugen. Das bedeutet, dass z.B. eine Auslenkung nach oben (z-Achse) als eine ebensolche Translation des virtuellen Objektes entlang der positiven z-Achse gemappt wird. Die hierfür verwendeten Potentiometer (Technik im aktuellen großen Kugelfisch) stammen von der Firma „Megatron“ und werden als MM10 bezeichnet. Sie sind in den Längen 8,11,12 und 15 cm verfügbar (Abb. 2.8). Außerdem weisen sie eine Linearitätsabweichung von 2% auf. In unserem Gerät kommen drei MM10 (für die drei Achsen) mit einer Länge von je 10 cm zum Einsatz.

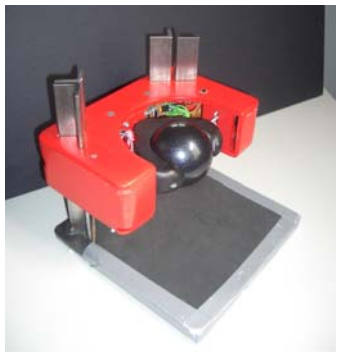

Abb. 2.5: Großer Kugelfisch.

Abb. 2.6: Kugel in der Fassung.

Abb. 2.7: Aufhängung der Fassung im Gehäuse.

Abb. 2.8: Potentiometer MM10.

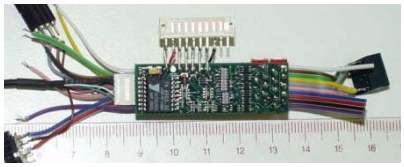

Abb. 2.9: Die VRIB-Inbox mit ihren Anschlussmöglichkeiten.(45 mm x 14 mm).

Bei den Potentiometern handelt es sich im Prinzip um Schiebewiderstände die mit einer so genannten VRIB-Inbox (Abb. 2.9) verlötet wurden.

Diese VRIB-Inbox ist ein A/D-Wandler, dessen Aufgabe darin besteht, die anlogen Signale der Potentiometer in digitale Werte umzuwandeln, um diese dann per USB an den Rechner weiterreichen zu können. Die VRIB-Inbox codiert die Auslenkung der Potentiometer in diskreten Werten zwischen 0 und 1024 (die Grenzen stehen für die Maximalauslenkung der Potentiometer, also die beiden Enden). Eine VRIB-Inbox ist in der Lage vier analoge und zehn diskrete Signale (Taster) zu wandeln und diese Informationen über einen USB-Anschluss weiter zu leiten. Die Stromversorgung erfolgt über den USB-Anschluss.

Kommen wir nun zur Fassung selbst, in der sich die Sensorik zur Abnahme der Kugelrotation befindet (Abb. 2.10). Die Kugel ist relativ frei zugänglich (Abb. 2.11) und lässt sich in um alle Achse frei rotieren. Um nun diese Rotation erfassen zu können befinden sich im Sockel zwei optische Sensoren, welche im 90° Winkel zueinander stehen und die Kugeloberfläche abtasten (Abb. 2.12). Auf der Suche nach geeigneter Sensorik boten sich optischen Sensoren aufgrund ihrer Funktionsweise und aus Kostengründen an. Für den Bau des großen Kugelfisches wurden optische Sensoren aus 2 handelsüblichen optischen Mäusen verwendet. Die im großen Kugelfisch verbauten optischen Sensoren stammen vom Modell „Marble Mouse USB MIN T- BC21“ der Firma „Logitech“ (http://www.logitech.com). Die Funktionsweise einer solchen optischen Maus und damit die ihrer optischen Sensoren soll hier kurz erklärt werden. Der optische

Abb. 2.10: Hier sieht man die optischen Sensoren, welche aus handelsüblichen optischen Mäusen entnommen wurden samt Platinen. Sie diene zur Erfassung der Rotation. (Das Bild zeigt ein ältere Entwicklungsstufe des großen Kugelfisches, beim aktuellen ist es jedoch ähnlich!)

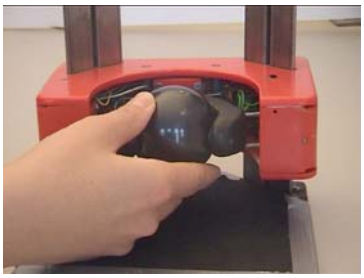

Abb. 2.11: Mögliche Handhaltung bei Manipulation der Kugel.

Sensor setzt sich im Prinzip aus drei Baugruppen zusammen: Einem Emitter (meist eine LED), einer Mini-Kamera und einem Mikrochip. Der Emitter sendet Licht auf die Unterlage auf die sich die Maus bewegt. Gleichzeitig nimmt die Kamera mit einer Frequenz von bis zu 1500 Hz die angestrahlte Unterlage auf und leitet die Bilder (16x16 Pixel) an den Mikrochip weiter. Dieser sucht in dem Bild nach Mustern. Hat er nun ein solches Muster gefunden, überprüft er ob und wenn ja

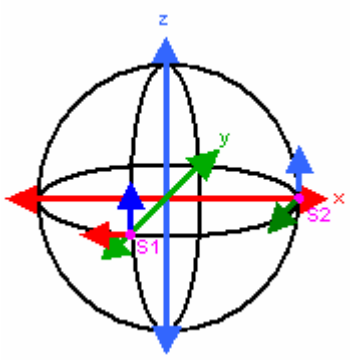

Abb. 2.12 :Anordnung der Sensoren und das dadurch aufgespannte Koordinatensystem.

wie sich dieses Muster vom Zeitpunkt n (repräsentiert durch das Bild) gegenüber dem Zeitpunkt n-1 (repräsentiert durch das vorhergehende Bild) verschoben hat. Konnte nun eine Verschiebung des Musters festgestellt werden, heißt das, dass sich die Maus bewegt haben muss. Diese Bewegung wird in einem 2D-Bewegungvektor codiert (x, y) und an den Rechner übermittelt, der dann diese Information (Bewegungsrichtung und Bewegungsstärke bzw. Länge) entsprechend interpretiert. Nun ist es in unserem Fall allerdings so, dass nicht der Sensor über die Unterlage gleitet, sondern die Unterlage (die Kugeloberfläche) an den Sensoren vorbei gleitet. Da wir allerdings Orientierungsänderungen im 3D erfassen müssen, benötigen wir zwei Sensoren, da einer nur in 2D auflösen kann. Da 2x2= 4 ist, haben wir nun sogar eine redundante Achse, somit können wir eine Achse doppelt messen, in unserem Fall die z-Achse. Dies stellt kein Problem dar, man kann z.B. beide z-Werte der Sensoren mitteln oder einfach einen weglassen. Die beiden Sensoren spannen nun im 90° Winkel zueinander angeordnet ein 3D-Koordienatensystem auf, wobei wie schon angesprochen die z-Achse doppelt vorhanden ist (Abb. 2.13).

Rotiert man die Kugel erhält man Inputs auf den optischen Sensoren, die mittels USB an den Rechner weitergeleitet werden. Somit müssen mindestens drei USB-Anschlüsse am Rechner zur Verfügung

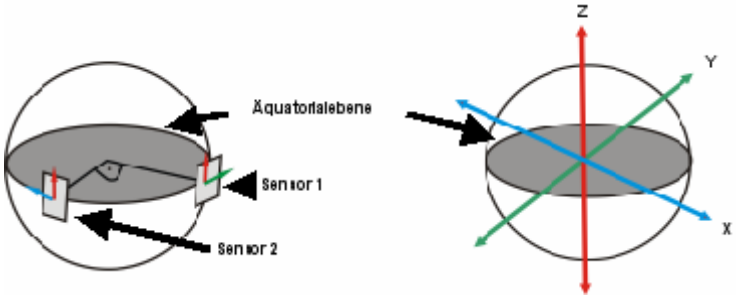

Abb. 2.13: Ausrichtung der Sensoren und die dadurch doppelte z-Achse.

stehen um das Gerät benutzen zu können (1x USB von VRIB-Inbox und je 1x USB von den optischen Sensoren).

Nun soll kurz auf technische und konstruktionsbedingte Probleme des Gerätes eingegangen werden. Als schwierig erwiesen sich die schwachen Rückzugfedern der Potentiometer, welche eigentlich dafür gedacht sind die Potentiometer nach der Auslenkung des Benutzers wieder in die Ausganglage (Mittelstellung, mathematischer Nullpunkt) zurück zu versetzen. Leider führten mechanische Defizite dazu, dass die Potentiometer nicht korrekt in die Ursprungslage zurückkehrten. Dadurch kam es auch ohne die Einwirkung des Benutzers dazu, dass eine Verschiebung an den Sensoren zu messen war. Diese Messwerte mussten im Programm abgefangen werden, um ein „Eigenleben“ der Anwendung bzw. des Eingabegerätes zu verhindern. Darüber hinaus gestaltet sich auch das Auslenken der Fassung als eine kraftaufwendige Handlung, doch dazu mehr wenn später die Auswertung der Probanden Fragebögen erfolgt.

Des weiteren gab es bei der Abnahme der Rotation Schwierigkeiten die korrekte Rotationsachse zu bestimmen, da aufgrund der Bauform eine Rotation immer an beiden Sensoren und somit an allen drei Inputs (x, y, z) Veränderungen hervorruft. Deshalb müssen die Rohwerte erst einige Prozeduren durchlaufen bevor mit ihnen gearbeitet werden kann.

Wie diese Probleme im Rechner mathematisch und programmtechnisch behandelt werden wird an späterer Stelle in dieser Dokumentation geklärt (Kapitel 5.4).

2.4. Kleiner Kugelfisch

Das Prinzip und der Aufbau des kleinen Kugelfisches ist das gleiche wie beim großen Kugelfisch. Er steht für eine Verbesserung des Kugelfischprinzips, aufgrund seiner anderen Sensorik für die Translation. Statt wie beim großen Kugelfisch Potentiometer, erledigt hier SpaceMouse-Technik das Erfassen der Verschiebung der Fassung. Die SpaceMouse-Sensorik ist auf der einen Seite in die Fassung eingelassen und dort fest mit dem Gehäuse verbunden. Auf der anderen Seite ist die Fassung lose im Gehäuse aufgehängt, um den notwendigen Raum für die Verschiebung bereitzustellen (Abb. 2.14, Abb. 2.15). Natürlich sind auch hier wieder zwei optische Sensoren in die Fassung eingelassen um die Rotation der Kugel abzugreifen. Auch diese zwei stammen von der „Marble Mouse USB MIN T-BC21“ von „Logitech“. Die Anordnung der ist die gleiche, wie beim großen Kugelfisch (Kapitel 2.3).

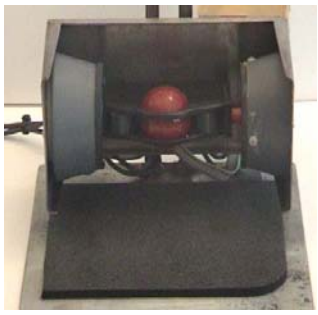

Abb. 2.14: Kleiner Kugelfisch.

In der linken Aufhängung befindet sich die SpaceMouse-Sensorik. In der rechten Aufhängung hat die Fassung Spiel um ausgelenkt zu werden

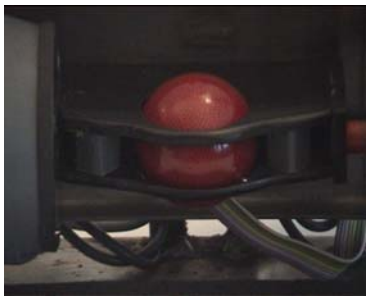

Abb. 2.15: Kleiner Kugelfisch. Kugel in der Fassung.

Ein Weiterer Unterschied zwischen beiden Kugelfischen, ist die Größe ihrer Kugeln. Im großen Kugelfisch kommt eine Kugel mit einem Durchmesser von 55 mm zum Einsatz. Beim kleinen Kugelfisch verrichtet eine kleinere Kugel (45 mm Durchmesser) ihre Arbeit. Diese Maßnahme dient der Untersuchung, ob die Kugelgröße Einfluss auf die Performances eines Gerätes hat.

Die SpaceMouse-Sensorik hat nicht das Defizit keine klare Nullstellung zu besitzen wie die Potentiometer, allerdings ist auch hier ein gewisser Kraftaufwand nötig um eine Auslenkung zu bewerkstelligen. Sicher hat dies mit der Art und Weise der Handhaltung bei der Bedienung des Gerätes zu tun. Doch dazu später bei der Fragebogenauswertung im Kapitel 6.3. mehr. Auch bei diesem Gerät gab es die beschriebenen Probleme bezüglichen der Detektion der korrekten Rotationsachse, doch auch hierzu später mehr (Kapitel 5.4).

2.5. Kugelmaus

Die Kugelmaus entstand als letztes Gerät. Es handelt sich dabei im Prinzip um eine SpaceMouse, die einen Kugelaufsatz bekommen hat (Abb. 2.16). Im Sockel verbirgt sich SpaceMouse-Technik, mit deren Hilfe die Verschiebung gemessen wird. Außerdem werden wieder zwei optische Sensoren verwendet, welche die darüber liegende im Sockel eingelassene Kugel abtasten (Abb. 2.17). Hier wurden optische Sensoren aus der Notebookmouse „NM 100“ von der Firma HAMA verwendet. Diese Sensoren stehen ebenfalls im 90° Winkel zueinander und spannen somit ein 3D-Koordinatensystem auf. Die Größe der Kugel beträgt 35 mm im Durchmesser. Die Handhaltung bei diesem Gerät entspricht in etwa der, bei der Bedienung einer üblichen Maus oder SpaceMouse. Man lenkt den Sockel aus um eine Verschiebung hervorzurufen (Abb. 2.18) und kann die Kugel drehen um eine Rotation zu produzieren (Abb. 2.19). Allerdings ist dies im Gegensatz zu den anderen Geräten nur sehr schwer simultan möglich, da bei der Kugelmaus die beiden Manipulationsarten eindeutig voneinander getrennt sind. Dies hat den Vorteil, dass ungewollte Überlagerungen der Manipulationsarten so gut wie unmöglich sind. Dies entspricht, wie im Kapitel 2.2 dargestellt, dem ersten Kritikpunkt an der SpaceMouse. Andererseits kann dies gleichzeitig ein Nachteil sein, da durch das Umgreifen effektive Arbeitszeit verloren geht. Ob sich diese Vermutung bewahrheitet, ist Teil der Auswertung der durchgeführten Versuchsreihen (Kapitel 6.2).

Abb. 2.16: Kugelmaus. Prinzip: SpaceMouse mit Kugelaussatz.

Abb. 2.17: Kugelmaus. Optische Sensoren (rechts und links) und Kugellager (Mitte).

Abb. 2.18: Kugelmaus. Manipulation am Sockel.

Abb. 2.19: Kugelmaus Manipulation an der Kugel.

2.6. Vergleichender Überblick

	SM	grKF	klKF	KM
Rotation	GS*	PS*	PS*	GS*
Translation	GS*	PS*	PS*	PS*
Rotations-Sensor	SpaceMouse-Sensorik-	Potentiometer	SpaceMouse-Sensorik	SpaceMouse-Sensorik
Translations-Sensor	SpaceMouse-Sensorik	optische Sensoren	optische Sensoren	optische Sensoren
Kugeldurch-messer	-	55 mm	35 mm	35 mm
Vorteile	• angenehme, vertraute Handhal-tung • simultane Benutzung, beider Steuerungs-modi	• intuitive RS* • Simultane Benutzung der Manipulati-onsarten be-dingt möglich	• intuitive RS* • Simultane Benutzung der Manipula-tionsarten be-dingt möglich • bessere Technik als beim grKF	• intuitive RS* • angenehme, vertraute Handhaltung • Manipulations-arten sind se-pariert
Nachteile	• häufig un-gewollte Simultane Benutzung der Manipu-lationsarten • wenig intui-tive RS*	• ungewohnte Handhaltung • schwergän-gige und feh-lerhafte Me-chanik	• ungewohnte Handhaltung • schwergän-gige Mechanik	• Umgreifen nö-tig
Anschlüsse	• serielle Schnittstelle (RS232)	• 3x USB	• serielle Schnittstelle (RS232) • 2x USB	• serielle Schnitt-stelle (RS232) • 2x USB

GS*: Geschwindigkeits-Steuerung, PS*: Positions-Steuerung

Abschließend noch eine schematische Darstellungen zu jedem einzelnen Gerät mit Manipulationsmöglichkeiten:

Abb. 2.20: SpaceMouse (klassik).

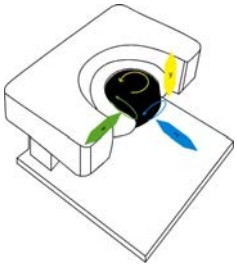

Abb. 2.21: Großer Kugelfisch.

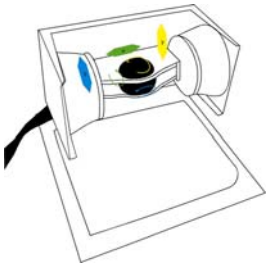

Abb. 2.22: Kleiner Kugelfisch.

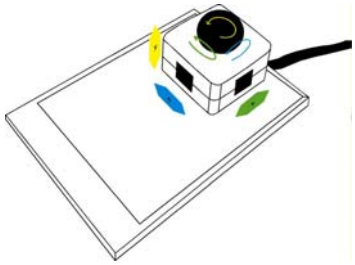

Abb. 2.23: Kugelmaus.

3. Hypothesen

Nun galt es, die an der BUW entwickelten Geräte, einem Vergleich mit der SpaceMouse zu unterziehen. Vor dem Vergleichstest stellten wir einige Hypothesen in die Raum, die es zu prüfen galt. Die wesentliche Idee der neuen Geräte, ist die Steuerung der Rotation mittels Manipulation einer Kugel. Diese Idee entstand aus der Überlegung heraus, wie man die wenig intuitive Rotationssteuerung der SpaceMouse besser, d.h. intuitiver realisiert werden könnte. Die logische Schlussfolgerung daraus war, dass die natürlichste Steuerung sicher diejenige wäre, bei der der Benutzer das Eingabegerät bzw. einen Teil davon genauso rotiert wie er auch gern das virtuelle Objekt rotieren würde. Somit fiel die Wahl auf eine Kugel als Manipulator. Daraus lässt sich also die erste zu überprüfende These ableiten:

Haupthypothese: Rotationen mittels Positionssteuerung einer Kugel sind intuitiver/einfacher zu steuern, als geschwindigkeitsgesteuerte Rotationen der SpaceMouse. Das Arbeiten gestaltet sich so bedeutend effektiver!

Nicht nur das Bedienungsprinzip des Eingabegerätes schlägt sich in dessen Performance nieder, sondern sicher auch die individuellen Unterschiede im Detail. Allen Geräten ist das Steuerungsprinzip mittels der Kugel gemein. Sie unterscheiden sich allerdings in der Kugelgröße. Somit ergibt sich die Frage ob die Kugelgröße Einfluss auf die Geräteperformance hat. Die 2. Hypothese lautet also:

Nebenhypothese-1: Die Kugelgröße hat Einfluss auf die Geräteperformance!

Wie schon bei der Vorstellung der Geräte erwähnt, ist die Kugelmaus das einzige Gerät, bei dem beide Manipulationsarten strikt getrennt sind. Dies verhindert natürlich das ungewollte simultane Ausführen der beiden Manipulationsarten, was für den ungeübten Benutzer ein schwer zu kontrollierender Effekt ist. Auf der anderen Seite allerdings macht die Bauweise des Eingabegerätes Umgreifen notwendig. Die Frage ist nun, ob dieser Nachteil den gewonnen Vorteil wieder wettmacht. Daher lautet die 3. Hypothese:

Nebenhypothese 2: Das Umgreifen bei der Kugelmaus mindert die Performance im Vergleich zu den Kugelfischen!

Um die aufgestellten Hypothesen zu überprüfen war es nötig, die Geräte in einem Test zu vergleichen. Im nächsten Abschnitt wird nun ausführlich auf den durchgeführten Test eingegangen.

4. EvaGrip2 Docking-Task: Versuchsmodalitäten

4.1 Versuchsdesign

Im Within-Subject-Design testeten 16 Versuchspersonen alle vier Geräte an zwei aufeinander folgenden Tagen, zwei Gräte pro Tag. Die Gerätereihenfolge wurde balanciert über die 16 Versuchspersonen im lateinischen Quadrat verteilt. Somit ergaben sich vier Gruppen mit jeweils vier verschiedenen Gerätereihenfolgen.

4.2 Versuchspersonen

Acht weibliche und acht männliche freiwillige Versuchspersonen nahmen an der EvaGrip2 Studie teil. Alle Teilnehmer waren rechtshändig und im Alter zwischen 21 und 26 Jahren und hatten sowohl 3D-Erfahrungen, als auch neben Tastatur und Maus schon mit anderen Eingabegerät gearbeitet bzw. gespielt.

Das Docking-Zielobjekt wurde von den Testpersonen in stereoskopischer Darstellung zwischen 8 cm und 16 cm vom Bildschirm entfernt wahrgenommen.

4.3 Ablauf einer Testreihe

Jede Testperson begann eine Testreihe zunächst mit drei Übungstrials, in der sie sich kurz mit der Aufgabenstellung vertraut machen konnte. Die Testreihe bestand aus vier Sessions, mit jeweils 12 Trials die aufgezeichnet wurden. Zwischen den Sessions wurde jeweils eine fünfminütiger Übungsphase geschaltet, die nicht protokolliert wurde. Eine Testreihe dauerte je nach Gerät etwa zwischen 45 und 60 Minuten.

4.4 Versuchsumgebung/Versuchsaufbau

Um allen Versuchspersonen die gleichen Bedingungen zu ermöglichen, muss die Versuchsumgebung gewissen Anforderungen genügen, damit die Testergebnisse von äußeren Einflüssen möglichst unberührt bleiben. Dabei war neben Ruhestörungen primär der visuelle Ablenkungsmoment zu verhindern, damit die Testperson während des Tests nicht abgelenkt oder gestört wird.

Abb. 4.1: Versuchsperson vor Stereo-Monitor, während Durchführung (Kugelmaus).

Der Versuchsraum wurde abgedunkelt, damit unabhängig von der Tageszeit gleiche Lichtverhältnisse gegeben waren. Neben Monitor und Tastatur befanden sich nur die aktuellen Testgeräte auf dem Versuchstisch an dem die Testperson allein saß. Der Versuchsleiter saß während der Testphase rechts hinter der Testperson, wie auf der Abbildung 4.1 zu sehen. Um einwandfreies stereoskopisches Sehen am Monitor zu gewährleisten, wurde die Bildschirmhalbierende auf Höhe der Sichtachse der jeweiligen Versuchsperson gebracht.

Utensilien im Testraum: Maßband, Stift, Fragebögen, Mineralwasser und Becher, Appetithäppchen.

4.5 Testaufgabe

Bei dem 6 DoF DockingTask von Shumin Zhai, sowie dem modifizierten Replik, wie es in EvaGrip2 zum Einsatz kam, besteht die Herausforderung darin, sowohl alle vorhandenen Freiheitsgrade einzusetzen, als auch von der Aufgabenstellung her einfach genug zu sein, um so die gängigsten 3D Applikation in virtuellen Umgebungen generalisieren zu können.

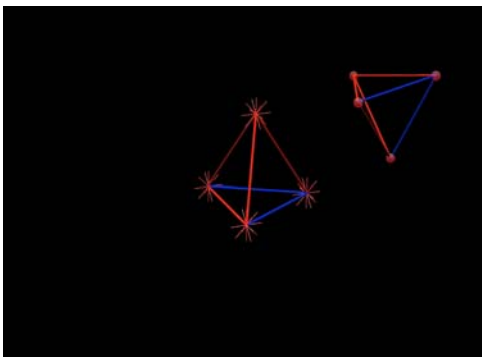

Abb.4.2: Target (Zentrum) und Cursor.

Die Versuchspersonen sollen so schnell wie möglich einen 3D Cursor mit einem 3D Target (Abb. 4.2) in Deckungsgleichheit bringen (Abb. 4.3): Cursor und Target sind Tetraeder von gleicher Größe und werden als Wireframe-Modell in schwarzer Umgebung dargestellt, d.h. die Strukturen im hinteren Teil des Tetraeders werden nicht von den vorderen Flächen verdeckt. Die Kanten der Tetraeder

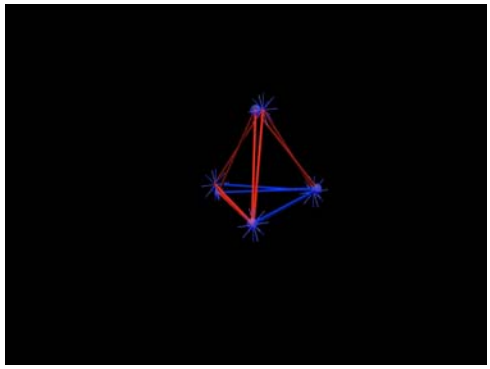

Abb.4.3: Erfolgreiches Docking.

sind mit verschiedenen Farben versehen, zwei Seiten sind blau und die restlichen vier Seiten sind rot gefärbt. Damit wird sichergestellt, dass es nur eine einzige Übereinstimmung in der Orientierung der Tetraeder zueinander gibt. Um Target und Cursor optisch unterscheiden zu können, besitzt der Target-Tetraeder an den Eckpunkten jeweils einen kleinen roten Stern, der Cursor-Tetraeder besitzt rote Kugeln.

Der Target-Tetraeder verbleibt während des gesamten Experiments in der Mitte des Bildschirms. Der Cursor-Tetraeder hingegen erscheint zu Beginn jedes Trials an einem von vier fest definierten Startpunkten in der Peripherie des Bildschirms. Alle Startpunkte im gleichen euklidischen Abstand zum Target, allerdings unterschiedlicher Orientierung. Eine Permutationsfunktion steuert, an welchem Startpunkt der Cursor jeweils auftaucht.

Tritt nun im Laufe des Trials ein Eckpunkt des Cursors in den dazugehörigen Toleranzbereich des Target-Eckpunkt ein, wird die Farbe des Sterns des Eckpunkts blau dargestellt. Damit wird angezeigt, dass dieser Eckpunkt richtig positioniert ist. Sind alle Eckpunkte für einen Zeitraum von mindestens 0.8 Sekunden richtig positioniert, gilt die Docking Aufgabe als erfüllt und die Task Completion Time (TCT) wird angezeigt.

Der Test wurde stereoskopisch mit einer Shutterbrille an einem 22'' Monitor mit einer Bildwiederholungsrate von ca. 48 Hz pro Auge dargestellt.

4.6 Fragebogendesign

Um die Hypothesen, die sich in der Eingangsfrage unseres Projekts ergaben, zu überprüfen wurde es erforderlich, die entwickelten Geräte durch Tests zu evaluieren. Ziel dieser Evaluation waren erste Ergebnisse über Ergonomie und Gebrauchsfähigkeit der Eingabegeräte, um die Hypothese der verbesserten Rotation mittels Manipulation einer Kugel nachzuweisen.

Wichtige Begriffe im wachsenden Wettbewerbsdruck sind Benutzungsfreundlichkeit, Gebrauchstauglichkeit, Benutzerfreundlichkeit, Benutzbarkeit und immer wieder Usability.

Uns interessierte bei dieser tatsächlichen Erhebung besonders die Umsetzung der Anforderung des Benutzers an eine einfache und intuitivere Bedienbarkeit, um so die Ergebnisse auf den Prototypen anzuwenden und eventuelle Unterschiede zur SpaceMouse herauszustellen.

Für die Bestimmung der Gebrauchsfähigkeit sind nach DIN EN ISO 9241-11 (Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten – Teil 11: Anforderungen an die Gebrauchstauglichkeit) die Eigenschaften Effektivität, Effizienz und Zufriedenheit zu gewährleisten. Während für die Effektivität Anforderungen an Exaktheit und Vollkommenheit der Arbeitsaufgabe wichtig sind, ist die Effizienz das Verhältnis der Effektivität an den benötigten Aufwand des Benutzers. Zufrieden ist dieser erst dann, wenn sich Akzeptanz, d.h. so wenig Störungen wie möglich, zwischen diesem und dem Eingabegerät ergibt.

Um genaue Angaben aus allen Empfindungsphasen der Testperson zu erhalten, war es notwendig 3 Fragebögen zu entwerfen. Da der Fragebogen das zentrale Verbindungsstück zwischen Theorie und Analyse ist, war es in dem **„Allgemeinen Fragebogen“** zunächst wichtig, die physiologischen Details der Testpersonen zu erfassen, auch um anzuzeigen, wie geübt oder ausgeprägt motorische Fähigkeiten der Testperson sind. Natürlich sollte der Allgemeine Fragebogen als Eingangsfragebogen auch die psychologische Aufgabe erfüllen, die Testperson vor allem als menschliche zu verstehen und sie mit so genannten Eisbrecherfragen anzuwärmen und zu interessieren. Dabei ergaben sich für uns folgende

Überlegungen: Wenn die Kugelgröße einen eventuellen Einfluss auf die Geräteperformance hat, dann ist es auch wichtig, wie groß die Hand ist – dazu erhoben wir als Maße die Länge der Hand (Wurzel bis Mittelfinger) und die Spannweite (Daumen bis Mittelfinger). Des Weiteren haben wir Gewohnheiten der Computernutzung auf den Fragebogen verzeichnet, um in etwa auszumachen, wie versiert und gewöhnt die Testperson bereits der Sinnesempfindung für 3D Umgebungen begegnet und in wie weit durch andere Beschäftigungen eine für Eingabegeräte authentische Fingerfertigkeit ausgebildet ist.

Aus diesem Grund fragten wir nach bereits anderen verwendeten Eingabegeräten neben Maus und Tastatur, so z.B. Joystick, oder Touchpad. Eine denkbare Gewöhnung der Testpersonen an 3D Umgebungen, zog die Frage nach sich, ob und wie oft darin von Ihr schon gespielt, bzw. gearbeitet wurde. Gedacht wurde dabei an Videospiele, 3D Simulationen, oder Arbeitsumgebungen wie z.B. Maya. Um die Feinmotorik der Testpersonen diffuser zu deuten, waren Erkenntnisse zu möglichen Aktivitäten wie Spielen eines Instruments, Handarbeit oder Modellbau nötig. Ferner um damit unsere Hypothese zu stützen, dass das Umgreifen bei der Kugelmaus die Performance in Gegenüberstellung zu den Kugelfischen vermindern würde.

In der anschließenden Testphase wurde nach jedem getesteten Gerät der **„Fragebogen Gerätetestphase“** gereicht, um die mit dem Gerät entstandenen Eindrücke als Spontanentwurf abzufangen. Auch hier ging es uns darum, zunächst einige allgemeine bis hin zu ganz spezifischen Fragen zu verfassen, die auch mit eigenen Worten beantwortet werden sollten. Da es und vor allem um die subjektiven Eindrücke der Testpersonen geht, verwendeten wir in allen Fällen eine numerische Skala mit verbalisierten Endpunkten. Es stellte sich uns, in der weiteren Fragebogengestaltung die Aufgabe der intervallskalierten Variablen. Bietet man die Gelegenheit der Mittelkategorie an, wird es möglich, dass die Testpersonen sie als Ausweichmöglichkeit nutzen, weil sie sich nicht auf das eine oder andere Profil der Skala einordnen wollen. Bedrängt man andererseits die Testperson, sich durch Auslassung der Mittelkategorie festzulegen, kann es passieren, dass die bewusste Entscheidung zur „Mittelmäßigkeit“ des Eingabegerätes fehlt und die Testperson sich gar nicht festlegt und so die Frage unbeantwortet auslässt. Da unsere Erhebung jedoch stark auf die Empfindsamkeit der Testperson und natürlich auch auf die der

Eingabegeräte gelenkt war, wurde es wichtig, zumindest ein tendenzielles Gutachten aller Testpersonen zu erhalten. In den Tests wurde schließlich auch keine Selbsteinschätzung verlangt, so dass die Mittelkategorie nur zum Nutzen des Sichnicht–festlegen-Wollens gedient hätte. Wir entschieden uns also für die Skala in 6 Punkten, die von sehr positiver zu sehr negativer Einschätzung hin abfallend geordnet war. Innerhalb der 6 Intervallpunkte würde es damit eine Erleichterung für die Versuchspersonen geben, da sie sich nicht konkret auf Gut oder Schlecht festlegen müssen, und anhand der 6 Punkte am ehesten Assoziationen zum Schulnoten-System knüpfen könnten, das ähnlich operiert: 1 für sehr gut und 6 für unzureichende Leistungen. Wichtige Einzelheiten zur Bewertung waren z.B. die Empfindlichkeit der Einstellung und die Umsetzung von der Handbewegung auf die Cursorsteuerung, bzw. die Größe der Kugel. Um alle Möglichkeiten der Eigenschaften zu erfassen, war es wichtig auch offene Fragestellungen zu formulieren, in denen die Testperson die Handhabe erhielt sich frei zu äußern und so zu Gefallen und Missfallen der Funktionen jedes Eingabegerätes befragt wurde.

Der Schluss-Fragebogen zur **„Endbewertung der Geräte“** sollte nachhaltig die Entscheidung der Versuchsperson beinhalten, sich für oder gegen eines der Eingabegeräte auszusprechen. Ausschlaggebend war für uns dabei den Favoriten auszumachen. Natürlich wollten wir auch dabei unsere Hypothese stützen, dass die Testpersonen sich gegen die Kugelmaus entscheiden würden, durch das ihr inhärente Umgreifen. Auch die Frage nach der Vergleichsweisen Beurteilung der Vor und Nachteile sollte die Testperson einladen, sich nochmals frei zu äußern und das mit den Eingabegeräten erlebte noch einmal zu reflektieren. Auch die Option der „weiteren Anmerkung“ war zu beachten, um ein eventuelles Feedback zu dem Fragebogen direkt oder zur Versuchsdurchführung zu erhalten.

4.7 Ablaufplan der Tests

In der Woche vom 01.07–07.07.2004 haben wir insgesamt 16 Versuchspersonen getestet. Um zu gewährleisten, dass die fünf Versuchsleiter die Testpersonen genau gleich behandeln, wurde vorher ein detaillierter Ablaufplan aufgestellt, der im Folgenden vorgestellt wird.

1) Bevor die Testperson eingetroffen ist:
   a) Gerät(e) anschließen und initialisieren.
   b) DockingTask starten und Gerät auf Funktion testen.
   c) Fragebögen in richtiger Reihenfolge geheftet bereitlegen.
   d) Versuchsraum abdunkeln.
2) Der Versuchsperson eine kurze Einführung/Erklärung geben:
   a) Um was es bei EvaGrip2 geht.
   b) Es werden 4 Geräte verglichen.
   c) Test bezieht sich auf die Geräte, nicht auf die Versuchspersonen.
   d) So schnell wie möglich die Tetraeder in Deckungsgleichheit zu bringen.
   e) Kinogutschein wird unter den Teilnehmern verlost.
3) Maße der VP im ***Fragebogen Allgemein*** eintragen:
   a) Handspanne.
   b) Daumen - Mittelfinger Spanne.
   c) Brillenträger.
   d) Linkshänder vermerken.
4) ***Fragebogen Allgemein*** von VP ausfüllen lassen.
5) Einstellungen:
   a) Sitz (Position und Höhe).
   b) Monitorabstand.
   c) Stereoskopietest (Abstand vom Monitor zum wahrgenommenen Cursor-Tetraeder notieren)

6) Task erklären:
   a) Starten des Tests mit Drücken der Space-Taste
   b) Deckungsgleichheit der Figuren (Cursor und Target).
   c) Lage der Achsen des Tetraeders (Rotation und Translation).
   d) Farbfeedback der Sterne am Target-Tetraeder bei richtiger Position.
   e) Dockingzeit: 0.8s.
7) Keine weiteren Erklärungen zu den Geräten, außer:
   a) SpaceMouse: Geräteausrichtung, wie wird verschoben und rotiert.
   b) Kugelfisch: getrennte Rotation und Translation (Benutzungshinweis).
   c) Kleiner Kugelfisch: nur Kugel anfassen, Höhe einstellen.
8) Bei jeder Session:
   a) Die Konfigurationsdatei **config.txt** anpassen (Session Nr., Gerät).
      i) "01" "name" 22 "abcd" "SM" "a" 1 "30.06.04_20.00" 0.
      ii) VP Nr.; Name; Alter; Gerätereihenfolge; Gerätname; Geräte-Flag; Session Nummer (1- 4); Datum; Übung (1) Test(0).
      iii) a: SpaceMouse, b: großer .Kugelfisch, c: Kugelmaus, d: Kleiner Kugelfisch
   b) vor der ersten Session nur 3 Trials, ansonsten 5 Minuten Übung.
      i) Hinweise zur Handhabung geben.
         (1) Gerät erproben bevor versucht wird zu docken.
         (2) fließende Bewegungen.
9) Start des Tests - Session (12 Trials)
   a) Wiederholung von Punkt 8 bis Session 4.

b) Nach Abschluss eines Testgerätes den ***Gerätespezifischen Fragebogen*** ausfüllen lassen.

10) Gerätewechsel

a) Punkt 8 und 9 wiederholen.

b) Nach Abschluss aller vier Geräte Fragebogen-Endbewertung ausfüllen lassen

5. Implementation

Die Zeit von Ende Mai 2004 bis Mitte Juli 2004 verbrachten wir überwiegend im Forschungslabor der Professur „Systeme der virtuellen Realität“ mit der Implementation unseres DockingTasks, welcher schon in Kapitel 4.5 näher beschrieben wurde.

5.1 Einleitung

In diesem Abschnitt wird etwas zum VR-System und den Anfängen in Scheme gesagt. Es wird erst der grobe Aufbau unseres Programms, die Modularisierung, Hinweise zu den Verzeichnisstrukturen, den Inhalten und den Zwecken unserer Scheme-Dateien erklärt. Es folgen genaue Angaben zu den Objekten der Szene und die Beschreibung der Funktionsweise von einzelnen wichtigen Teilstücken des Programms.

Das Programm ist dynamisch und modularisiert. Dies bedeutet, dass die richtigen Dateien automatisch geladen und ausgeführt werden. Das Öffnen, Schreiben und Abspeichern unter automatisch generierten Dateinamen läuft selbstständig ab.

5.1.1 Das VR-System Avango

Auf den Rechnern im VR-Lab ist die Linux Distribution Red Hat installiert. Den Nachbau von Zhai’s DockingTask realisierten wir mit der Scriptsprache Scheme, die das VR-System Avango steuert. Avango wurde am heutigen Fraunhofer Institut entwickelt.

5.1.2 Erste Anfänge in Scheme

Anfänglich mussten wir uns mit der Scriptsprache Scheme und dem VR-System vertraut machen. Ebenso mit der Einbindung und der Handhabung der verschiedenen Eingabegeräte. Wir konnten auf einige schon vorhandene Implementierungen zugreifen und haben so schnell die Funktionsweise von Scheme verstanden.

Unser Konzept sah anfänglich vor, für jedes Gerät ein vollständiges eigenes Programm zu erstellen. Doch da dies bedeutet, dass viele Teile des Codes, wie zum Beispiel die Startfiles und der Callback, mehrfach vorhanden wären, haben wir uns im Laufe der Programmierarbeit entschlossen ein modularisiertes Programm zu entwickeln, welches nur die gerätespezifischen Codestücke einbindet während der restliche Part immer derselbe ist.

5.2 Grober Ablauf und Aufbau des Programms

Um einen ersten Überblick über das Programm und seinen Aufbau zu erhalten, beschreiben wir zunächst, die Punkte, die der Versuchsleiter durchführt, um das Programm zu starten, sowie den internen Programmablauf.

Versuchsleiter

1. startet den jeweiligen Gerätedaemon in einer Shell. Im Falle des Kugelfisches wäre dies ./start-daemon_klK.
2. die Datei config.txt muss angepasst werden.
3. das Hauptprogramm wird mit ./start-stereo-tetra in einer zweiten Shell aufgerufen.

Intern im Programm laufen nachstehende Schritte ab:

(Auf die genaue Wirkweise von Funktionen und Flags, sowie Erläuterungen von einzelnen Codestücken wird später noch eingegangen):

1. ./start-stereo-tetra ruft das Hauptprogramm “evagrip2-zhai-replikation-tetraeder.scm” auf.
2. scripts/config.txt wird geöffnet, die Angaben ausgelesen und in Variablen gespeichert.
3. scripts/mouse-ptr.scm wird geladen und der Mauszeiger ausgeblendet.
4. Die Geometrien werden geladen (load "scripts/geometry.scm").
5. Sie beschreiben den Aufbau der Szenerie, definieren die Grenzen des Raumes, Lichtquellen, Cursor- und Target-Tetraeder sowie die Eckmarkierungen von beiden.

6. Notwendige Variablen und Flags werden definiert und gesetzt.
7. scripts/display.scm wird geladen. In dieser Datei sind die Informationstexte beschrieben, die während des Programmablaufs dargeboten werden. Z.B. die Dockingzeit und die Aufforderung den nächsten Trial zu starten.
8. scripts/functions_and_writeout.scm wird geladen; diese Datei definiert folgende Funktionen, die im Kapitel 5.3.8 noch näher erläutert werden.
    i. BoundingBoxFunction (verhindert, das der Cursor den Bildschirmbereich verlässt),
    ii. WriteOutFunctionHeader,
    iii. WriteOutFunction,
    iv. WriteOutToleranz,
    v. ListPickFunction ist eine kleine Hilfsfunktion für StartPosChoise. Sie wählt immer das n-te Element aus einer list of number (lon) , und setzt dieses der Variable _listenelement gleich.
    vi. StartPosChoiseTrials und
    vii. StartPosChoiseUebung
9. Dann wird, abhängig vom DeviceTypeFlag, das jeweilige Gerätescript geladen.
10. Die Scripts sind zwar für jedes Gerät verschieden, aber sie haben alle den selben Aufbau und Zweck: Sie kümmern sich darum, dass ...
    1. ... die Rohwerte, die von den Geräten geliefert werden, korrekt auf CursorRot und CursorTrans multipliziert werden.
    2. ... die Eckpositionen ihre Farbe wechseln, sobald eine Ecke im Toleranzbereich ist.
    3. ... die Szene eingefroren wird, sobald der Cursor länger als 0.8 s mit allen Ecken innerhalb der Toleranz ist.
    4. ... die Informationstexte ein- und ausgeblendet werden.
    5. ... ganz am Anfang callback.scm geladen wird
11. Im Callback wird als erstes die Unterscheidung zwischen Übung und Datenerhebung vorgenommen und die jeweilige StartPosChoise-Funktion

aufgerufen. Anschließend werden Zeiten gesetzt und die Datei gewählt, in welche die Daten geschrieben werden sollen. Die Header werden in die Files geschrieben und die Writeoutfunction (mit verschiedenen Parametern für jedes Gerät) aufgerufen. Die BoundingBox wird aufgerufen, und die Ecken der Tetraeder bekommen ihre aktuelle Farbe. Des weiteren werden noch ein paar Flags und Zeiten gesetzt.

5.3 Feinheiten der Programmierung

Im vorangegangenen Abschnitt haben wir kurz den Ablauf des Programms geschildert. Nun geht es um einige ausgewählte wichtige Teilstücke unseres Codes, wie:

5.3.1. Config-Datei
5.3.2. Geräteeinbindung (Gerätedaemon und Events)
5.3.3. Szenerie
5.3.4. Translation & Rotation der Objekte
5.3.5. Cursorstartpositionen
5.3.6. Stereo & ScaleDCS
5.3.7. Toleranz und Distanzvektoren
5.3.8. Datenaufzeichnung, Functions & Writeout
5.3.9. Aufrufe und Implementierung im Callback.scm
5.3.10. Hinweise zu den Verzeichnisstrukturen

5.3.1 Die config-Datei:

Die erste Zeile der config.txt sieht z.B. so aus:

```
0"0" "name" 21 "abcd" "klKF" "d" 1 "30.06.04_20.00" 0
```

Dabei haben die Angaben folgende Bedeutung:

Eintrag	`Variablenname` und Bedeutung
“01”	`PersonalNumber` Die Versuchspersonennummer hat eine führende Null (“01”, bzw. “12”) und wird bei der Dateinamengenerierung verwendet.

Eintrag	`Variablenname` und Bedeutung
“name”	`TestSubjectName` Nachname der Versuchsperson wird ebenfalls bei der Dateinamengenerierung verwendet.
21	`TestSubjectAge` Das Alter wird ohne “Anführungsstriche” angegeben, da es einen numerischen Wert repräsentiert. Diese Angabe ist nur für statistische Zwecke wichtig.
“abcd”	`DeviceOrder` Dieser String beschreibt die Gerätereihenfolge. Sie wird durch das lateinische Quadrat der vier Buchstaben a, b, c und d beschrieben, kann also mit „abcd“, "bcda“, „cdab“ oder dabc“ angegeben werden. a steht hierbei für die SpaceMouse, b für den großen Kugelfisch, c für die Kuglemaus und d für den kleinen Kugelfisch.
“klKF”	`DeviceType` Kürzel des Gerätes, wichtig für Dateinamengenerierung.

“d”	`DeviceTypeFlag`
	
	Dies ist der Geräteflag. Er ist extrem wichtig für den load-Befehl der richtigen Gerätedatei. Anhand der Buchstaben a, b, c und d wird die richtige Gerätedatei geladen und ausgeführt.
	`(if (equal? DeviceTypeFlag "a")`
	`(begin`
	`(display "SM") (newline)`
	`(load "scripts/SpaceMouse.scm")))`
	`(if (equal? DeviceTypeFlag "b")`
	`(begin`
	`(display "grKF")(newline)`
	`(load "scripts/grKugelfisch.scm")))`
	`(if (equal? DeviceTypeFlag "c")`
	`(begin`
	`(display "KM") (newline)`
	`(load "scripts/kugelmouse.scm")))`
	`(if (equal? DeviceTypeFlag "d")`
	`(begin`
	`(display "klKF")(newline`
	`(load "scripts/klKugelfisch.scm")))`

Eintrag	`Variablenname` und Bedeutung
1	`Session` Diese numerische Variable kann den Wert 1, 2, 3, oder 4 annehmen. Anhand von ihm wird die richtige Session aufgerufen und somit die richtigen Startpositionen gewählt.
“30.06.04_20.00”	`Date` Die Angabe von Datum und Uhrzeit ist optional, sie wird aber im Header der Dateien mit herausgeschrieben.
0 oder 1	`UebungsFlag` 0 oder 1, das ist hier die Frage. Wird eine 1 angegeben, wird die Übung ausgeführt, wird eine 0 angegeben, erfolgt die Datenerhebung. Bei der Übung werden 5 Minuten Trials dargeboten, bei der Datenerhebung jeweils nur 12 Trials.

5.3.2 Geräteimplementierung

In diesem Unterkapitel wird exemplarisch mit dem kleinen Kugelfisch auf die Geräteeinbindung in Avango eingegangen, da dieser sowohl die Spacemouse, als auch die optischen Sensoren enthält.

Für die Verwaltung eines Eingabegerätes wird unter Linux ein Dämonprozess gestartet, der av-deamon, welcher in einer eigenen Shell als Hintergrundprozess läuft und die Signale der Geräte verarbeitet. Ein Startskript startet diesen und stellt dann zusammen mit *av-linux-event.scm* den gerätespezifischen Skripten die Variablen der Geräte-Events zur Verfügung.

Um nun beispielsweise den kleinen Kugelfisch in Avango einzubinden, wird das Startskript *./start-daemon_klKF* gestartet.

```
#! /bin/sh -f
export AVANGO_CONSOLE_FILE_NAME=./log
../avango/bin/av-daemon.sh -f scripts/devices_klK.scm $*
```

Da Linux die USB- Belegung der Geräte dynamisch vergibt und einige Komponenten der Eingabegeräte eine USB- Schnittstelle haben, muss man ggf. mit *./ev_read*

/dev/input/eventX (X = 0, 1, 2, 3, ...) die ankommenden Geräte-Events abfangen und in dem jeweiligen Geräteskript anpassen.

Hier ein Beispiel für *device_klK.scm* :

```
…
(define usb2 (add-device    "usb-hid2"   "fpHIDInput"))
(config-device "usb-hid2"   "tty" "/dev/input/event5")
…
```

Wird dies versäumt, kann es sein, dass die ankommenden Signale des Gerätes nicht oder falsch gemappt werden und Translation sowie Rotation nicht richtig ausgeführt werden.

```
…
(add-device "spacemouse" "fpSpaceMouse")
(config-device "spacemouse" "tty" "/dev/ttyS0")
(config-device "spacemouse" "baud-rate" "9600")
(add-station "spacemouse" "spacemouse" 1)
(start-device "spacemouse")
…
```

An dieser Stelle wird durch *add-device “spacemouse“* die eingebaute SpaceMouse in dem kleinen Kugelfisch vom Typ “fp-SpaceMouse“ hinzugefügt und der ersten seriellen Schnittstelle ttyS0 zugewiesen.

Der *device-service* wird einmalig definiert und mit dem daemon verknüpft. Außerdem wird für jedes Eingabegerät ein Sensor *"fpADSensor"* benötigt, der später die sich ständig ändernden Werte des Eingabegerätes zurückgibt. Nun können die Werte wie folgt in der Applikation verwendet werden.

```
…
(define device-service (make-instance-by-name "fpDeviceService"))
(-> device-service 'register-service "Device-Service")
(-> device-service 'connect-daemon)

(define SpaceMouse (make-instance-by-name "fpADSensor"))
(fp-set-value SpaceMouse 'Station "spacemouse")

(define fish-sensor2 (make-instance-by-name "fpADSensor"))
(fp-set-value fish-sensor2 'Station  "usb-station2")

(define fish-sensor3 (make-instance-by-name "fpADSensor"))
(fp-set-value fish-sensor3 'Station  "usb-station3")
…
```

Mit diesen Definitionen sind die SpaceMouse und die optischen Sensoren in Avango verfügbar und können z.B. mit *(fp-dump spacemouse)* oder mit *(fp-dump usb-station2)* ausgelesen werden.

Über die callbackfunction *fetchCB* werden die Rohwerte der Geräte mit (fp-get-value SpaceMouse 'Value0) ausgelesen und in Variablen (z.B. *temp*_value*X*) geschrieben. Bei einigen Achsenwerten der SpaceMouse und der optischen Sensoren können bei der Auslenkung der Verschiebung bzw. bei der Rotation antiproportionale Werte ankommen, die dann durch die Multiplikation mit -1 invertiert werden.

```
…
(define (fetchCB value)
      (let (
            (_tmp_valueX (* (fp-get-value SpaceMouse 'Value2) 1))
            (_tmp_valueY (* (fp-get-value SpaceMouse 'Value0) 1))
            (_tmp_valueZ (* (fp-get-value SpaceMouse 'Value1)  1))
            (_f2x   (* (fp-get-value fish-sensor2 'Value0) -1))
            (_f2y   (* (fp-get-value fish-sensor3 'Value0) 1))
            (_f3y   (* (fp-get-value fish-sensor2 'Value1) 1))
            (_f3x   (* (fp-get-value fish-sensor3 'Value1) 1))
            (Spaceba_tmp_value (fp-get-value av-input 'SpaceKey))
            (Button0 (fp-get-value av-input 'AltKey))
)
…
```

Um nun die jeweiligen sechs Achsenwerte bei der Auslenkung zu erfahren, d.h. welche Bewegung der SpaceMouse welchem x,y,z- Wert der Translation bzw. welchen rx,ry,rz- Wert der Rotation entspricht, lässt man sich die Werte der Values 0-5 jeder SpaceMouse ausgeben.

```
…
(define (fetchCB value)
(display „v0: „) (display (fp-get-value SpaceMouse ’Value0)
(display „v1: „) (display (fp-get-value SpaceMouse ’Value1)
(display „v2: „) (display (fp-get-value SpaceMouse ’Value2)
(display „v3: „) (display (fp-get-value SpaceMouse ’Value3)
(display „v4: „) (display (fp-get-value SpaceMouse ’Value4)
(display „v5: „) (display (fp-get-value SpaceMouse ’Value5)
...
```

Alle Maximalwerte der Freiheitsgrade werden dann auf 1 normiert.

Hier exemplarisch die Normierung des kleinen Kugelfisch:

```
…
(if (> _tmp_valueX 0) (begin (set! _tmp_valueX (* _tmp_valueX (expt 0.82 -
1)))))
(if (< _tmp_valueX 0) (begin (set! _tmp_valueX (* _tmp_valueX (expt 0.7 -
1)))))

(if (> _tmp_valueY 0) (begin (set! _tmp_valueY (* _tmp_valueY (expt 0.45 -
1)))))
(if (< _tmp_valueY 0) (begin (set! _tmp_valueY (* _tmp_valueY (expt 0.55 -
1)))))

(if (> _tmp_valueZ 0) (begin (set! _tmp_valueZ (* _tmp_valueZ (expt 1.4 -
1)))))
(if (< _tmp_valueZ 0) (begin (set! _tmp_valueZ (* _tmp_valueZ (expt 0.53 -
1)))))
```

```
...
```

Da die SpaceMouse sehr empfindlich ist und allein von der Gravitationskraft schon ausgelenkt wird, wurde eine Schwellenwerte für Translation definiert, die durch Austesten entstanden sind.

```
…
(define _trans_threshold 0.075)
(define _transScale 0.15)
...
```

Sind nun die absoluten Eingangswerte des Gerätes kleiner als der Schwellwert, wird der Wert auf 0 gesetzt und keine Bewegung ausgelöst. Ist der Eingangswert größer als der Schwellwert, wird die Translation normal ausgeführt.

```
...
(if (< (abs _tmp_valueX) _trans_threshold)
      (begin  (set! _tmp_valueX 0.0))
      (begin  (if (> _tmp_valueX 0) (begin (set! _tmp_valueX (* (-
_tmp_valueX _trans_threshold) (expt (- 1 _trans_threshold) -1)))))
            (if (< _tmp_valueX 0) (begin (set! _tmp_valueX (* (+
_tmp_valueX _trans_threshold) (expt (- 1 _trans_threshold) -1)))))
      ))

(if (< (abs _tmp_valueY) _trans_threshold)
      (begin  (set! _tmp_valueY 0.0))
      (begin  (if (> _tmp_valueY 0) (begin (set! _tmp_valueY (* (-
_tmp_valueY _trans_threshold) (expt (- 1 _trans_threshold) -1)))))
            (if (< _tmp_valueY 0) (begin (set! _tmp_valueY (* (+
_tmp_valueY _trans_threshold) (expt (- 1 _trans_threshold) -1)))))
      ))

(if (< (abs _tmp_valueZ) _trans_threshold)
      (begin  (set! _tmp_valueZ 0.0))
      (begin  (if (> _tmp_valueZ 0) (begin (set! _tmp_valueZ (* (-
_tmp_valueZ _trans_threshold) (expt (- 1 _trans_threshold) -1)))))
            (if (< _tmp_valueZ 0) (begin (set! _tmp_valueZ (* (+
_tmp_valueZ _trans_threshold) (expt (- 1 _trans_threshold) -1)))))
      ))
…
```

Die temporären Gerätewerte werden dann mit 3 potenziert und kontinuierlich auf die Matrixwerte addiert, sodass die Funktion 3. Grades bei stärkeren Auslenkungen schneller Bewegungen der Translation, bei kleinen Auslenkung dementsprechend kleinere Bewegungen der Translation hervorrufen.

```
…
(fp-set-value CursorTrans 'Matrix(mult-mat
      (fp-get-value CursorTrans 'Matrix) )
      (make-trans-mat
      (* _transScale (expt _tmp_valueX 3))
      (* _transScale (expt _tmp_valueY 3))
      (* _transScale (expt _tmp_valueZ 3)))
))
```

Zur Implementierung der Rotation durch die optischen Sensoren in dem Kugelfisch und der Kugelmaus wird ausführlich in Kapitel 5.4 eingegangen.

5.3.3 Die Szenerie

In der Szene befinden sich 2 graphische Objekte, der Cursor und das Target. Beide sind, wie in Kapitel 4.5 beschrieben, Wireframe Tetraeder, die an ihren Ecken Markierungen haben.

Da die Ecken ihre Farbe wechseln, mussten Target und Cursor aus dem Tetraeder und 4 blauen und 4 roten Markierungen zusammengesetzt und “gruppiert” werden. Soll dann z.B. ein blaues Sternchen beim Target dargeboten werden, ist das jeweilige rote Sternchen ausgeblendet.

Der Wireframe Tetraeder sowie die Sternchen sind in Maya entstanden und als InventorFile (*.iv) exportiert. In Maya ist der Koordinatenursprung gleichzeitig der Schwerpunkt eines Objektes, um den in Avango rotiert wird. Zu Beginn lag der geometrische Schwerpunkt des Tetraeders nicht im Koordinatenursprung. Dies hatte zur Folge, dass der Tetraeder um einen Punkt auf einer Seite rotierte und man den Eindruck hatte, als ob er “eiert”. Bei der Modellierung in Maya kam es also darauf an, dass jede Ecke des Tetraeders den gleichen Abstand d (distance) zum Ursprung hat. Diese Strecke lässt sich mit der Formel

```
d = (sqrt6)/4)*Seitenlänge Tetraeder
```

berechnen. Bei einer Seitenlänge von 1 LE, beträgt dieser Abstand rund 0.61 LE.

Wird in Avango ein Objekt angelegt, entsteht dieses im Koordinatenursprung, wenn keine Angaben über eine Translation gemacht werden. Die Koordinaten A (0, 0, 0.605), B (-0.289, 0.494, -0.206), C (0.566, 0, -0.206) und D (-0.289, -0.494, -0.206) wurden von Hand berechnet. Sie entsprechen den Eckpunkten und haben alle eine Entfernung von rund 0.61 LE zum Schwerpunkt des Tetraeders. An diese Positionen wurden die Markierungen verschoben und dann als Kinder mitsamt dem Tetraeder an das Target bzw. den Cursor gehängt.

```
(define _Ax  0.00)
(define _Ay  0.00)
(define _Az  0.605)
...

(define Cursor-tetra (make-instance-by-name "fpLoadFile"))
```

```
(fp-set-value Cursor-tetra 'Name "Cursor-tetra")
(fp-set-value Cursor-tetra 'Filename
"geometry/tetraeder_scaled_colored_slim_centered.iv")
...

(define Kugel-rot-1 (make-instance-by-name "fpLoadFile"))
(fp-set-value Kugel-rot-1 'Filename "geometry/sphere_rot.iv")
(fp-set-value Kugel-rot-1 'Name "Kugel-rot-1")
(fp-set-value Kugel-rot-1 'Matrix (make-scale-mat 0.05 0.05 0.05))
(fp-set-value Kugel-rot-1 'DrawMask -1)
...

(define Cursor-ecke-1 (make-instance-by-name "fpDCS"))
(fp-set-value Cursor-ecke-1 'Name "Cursor-ecke-1")
(fp-set-value Cursor-ecke-1 'Children(list Kugel-rot-1 Kugel-blau-1))
(fp-set-value Cursor-ecke-1 'Matrix (make-trans-mat _Ax _Ay _Az))
....
(define Cursor (make-instance-by-name "fpDCS"))
(fp-set-value Cursor 'Name "Cursor")
(fp-set-value Cursor 'Children
            (list Cursor-tetra
                  Cursor-ecke-1
                  Cursor-ecke-2
                  Cursor-ecke-3
                  Cursor-ecke-4))
```

Da das Target keinen guten räumlichen Tiefeneindruck bot, wurde es noch um jeweils 20° um die X und die Y Achse rotiert, so dass die Kanten sich nicht mehr gegenseitig überlagerten, und so ein besserer Tiefeneindruck entstand. Zusätzlich wurde das Target noch um 2 LE in Richtung der negativen Y-Achse verschoben:

```
(fp-set-value TargetRot 'Matrix (mult-mat (make-rot-mat 20 0 0 1)
                                          (make-rot-mat 20 1 0 0)))
 (fp-set-value TargetTrans 'Matrix (make-trans-mat 0 -2 0))
```

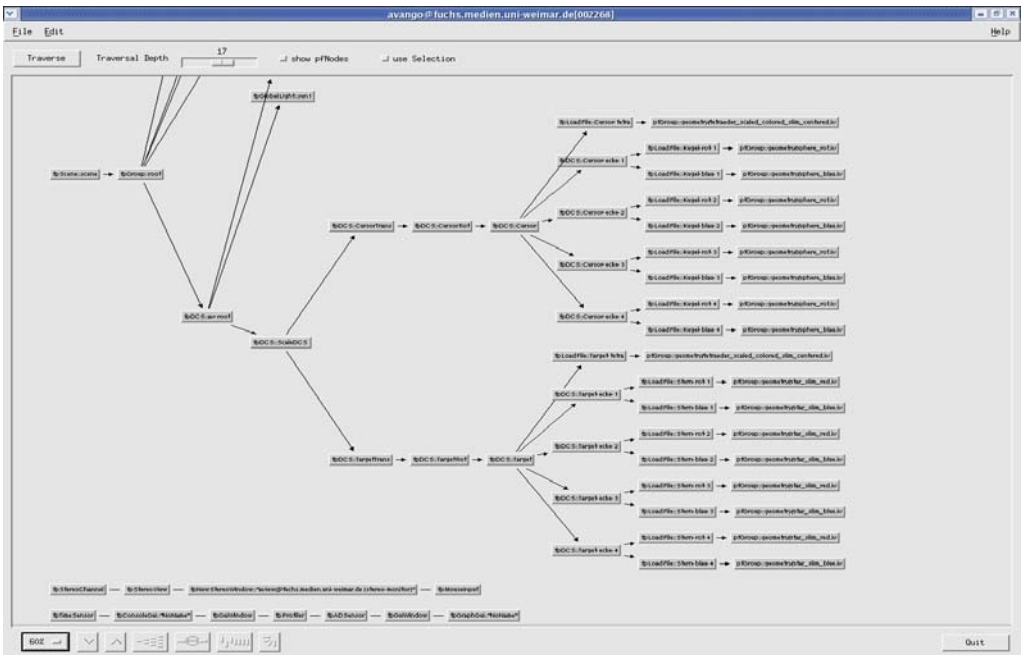

Abb. 5.1: Screenshot Szenengraph.

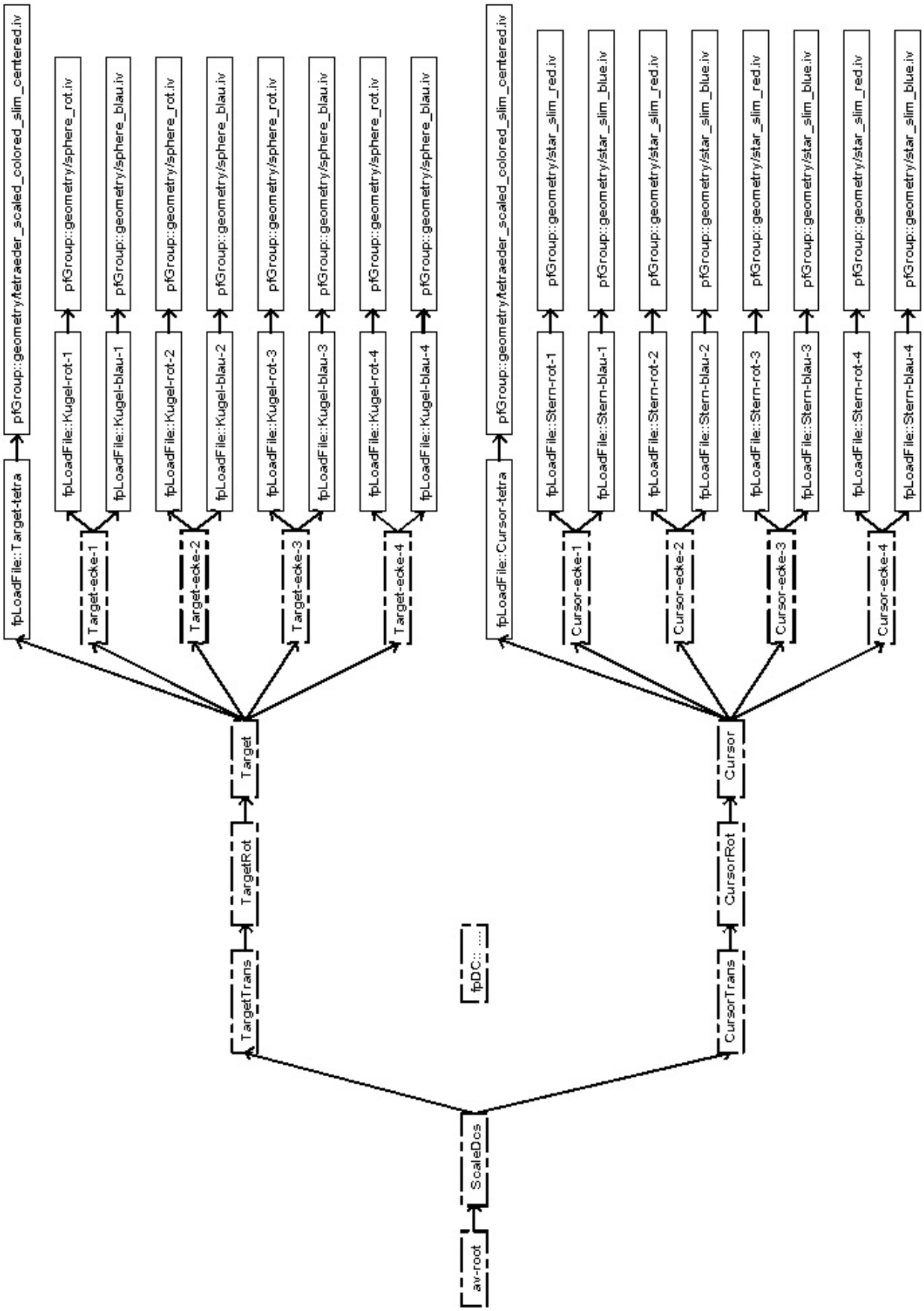
av-root
ScaleDcs
fpDC:: ....
TargetTrans
TargetRot
Target
fpLoadFile::Target-tetra
pfGroup::geometry/tetraeder_scaled_colored_slim_centered.iv
Target-ecke-1
fpLoadFile::Kugel-rot-1
fpLoadFile::Kugel-blau-1
pfGroup::geometry/sphere_rot.iv
pfGroup::geometry/sphere_blau.iv
Target-ecke-2
fpLoadFile::Kugel-rot-2
fpLoadFile::Kugel-blau-2
pfGroup::geometry/sphere_rot.iv
pfGroup::geometry/sphere_blau.iv
Target-ecke-3
fpLoadFile::Kugel-rot-3
fpLoadFile::Kugel-blau-3
pfGroup::geometry/sphere_rot.iv
pfGroup::geometry/sphere_blau.iv
Target-ecke-4
fpLoadFile::Kugel-rot-4
fpLoadFile::Kugel-blau-4
pfGroup::geometry/sphere_rot.iv
pfGroup::geometry/sphere_blau.iv
CursorTrans
CursorRot
Cursor
fpLoadFile::Cursor-tetra
pfGroup::geometry/tetraeder_scaled_colored_slim_centered.iv
Cursor-ecke-1
fpLoadFile::Stern-rot-1
fpLoadFile::Stern-blau-1
pfGroup::geometry/star_slim_red.iv
pfGroup::geometry/star_slim_blue.iv
Cursor-ecke-2
fpLoadFile::Stern-rot-2
fpLoadFile::Stern-blau-2
pfGroup::geometry/star_slim_red.iv
pfGroup::geometry/star_slim_blue.iv
Cursor-ecke-3
fpLoadFile::Stern-rot-3
fpLoadFile::Stern-blau-3
pfGroup::geometry/star_slim_red.iv
pfGroup::geometry/star_slim_blue.iv
Cursor-ecke-4
fpLoadFile::Stern-rot-4
fpLoadFile::Stern-blau-4
pfGroup::geometry/star_slim_red.iv
pfGroup::geometry/star_slim_blue.iv

Abb. 5.2: Ausschnitt aus dem Szenengrafen.

5.3.4 Translation & Rotation der Objekte

Rotation und Translation der Tetraeder werden getrennt voneinander behandelt. Für jede Transformationsart gibt es pro Objekt einen eigenen Knoten im Graphen. Das liegt daran, dass man die Transformationen nacheinander ausführen muss. Erst muss ein Objekt transliert, dann rotiert werden.

Dass es auf die Reihenfolge der Transformationen ankommt, lässt sich wie folgt zeigen:

A sei die Ausgangsmatrix, R die Rotationsmatrix, T die Translationsmatrix und E die Einheitsmatrix

Annahme: A*T*R=A*R*T

Gegenbeweis:

Da die Matrizenmultiplikation nicht kommutativ ist, gilt: T*R != R*T

(Einzige Ausnahme: T und R sind invers zueinander und ergeben in der Multiplikation die Einheitsmatrix E: T*R=R*T=E.)

→T*R=C

→R*T=D,

→A*C!=A*D, Widerspruch zur Annahme, d.h. A*T*R!=A*R*T

5.3.5 Cursorstartpositionen

Der Cursor hat 4 Startpositionen, an denen er in zufällig erscheinender Reihenfolge dargeboten wird. Da die Vergleichbarkeit zwischen einzelnen Trials bei den Versuchspersonen gegeben sein soll, ist die Reihenfolge für jede Person immer die gleiche:

```
(if (= UebungsFlag 0)
      (begin
            (define _session1_list (list 1 2 3 4 2 4 1 3 3 1 4 2))
            (define _session2_list (list 4 1 3 2 2 3 1 4 3 4 2 1))
            (define _session3_list (list 1 4 3 2 4 2 1 3 2 3 4 1))
            (define _session4_list (list 4 3 2 1 3 1 2 1 1 3 2 4))
      )
)
```

Die Zahlen eins bis vier symbolisieren folgende Startpositionen des Cursors:

Startposition 1: (-2.5, 0, 1.5), Startposition 2: (2.5, 0, 1.5),

Startposition 3: (-2.5, 0, -1.5). Startposition 4: (2.5, 0, -1.5)

```
(define _start_rot_1    (mult-mat   (make-rot-mat -110 0 1 0)
                                    (make-rot-mat 60 0 0 1)
                                    (make-rot-mat 20 1 0 0)))

(define _start_trans_1       (make-trans-mat  -2.5  0 1.5))


(define _start_rot_2    (mult-mat   (make-rot-mat -110 0 1 0)
                                    (make-rot-mat -60 0 0 1)
                                    (make-rot-mat 20 1 0 0)))

(define _start_trans_2       (make-trans-mat 2.5 0 1.5))

(define _start_rot_3 _start_rot_2)
(define _start_trans_3  (make-trans-mat -2.5 0 -1.5))

(define _start_rot_4 _start_rot_1)
(define _start_trans_4 (make-trans-mat 2.5 0 -1.5))
```

Die Startpositionen haben alle die gleiche euklidische Distanz von 12.5 LE zu der Targetposition (0, -2, 0) und sind ähnlich rotiert worden.

5.3.6 ScaleDCS

Befindet man sich im Stereo-Modus, so gelten für die Objekte noch andere ReferenzKoordinaten. Von Avango wird ein Koordinatensystem aufgebaut, welches seinen Ursprung im Zentrum des Bildbereiches liegen hat. Die X-Achse verläuft von links (negativ) nach rechts (positiv), die Z-Achse wie gewohnt von unten (negativ) nach oben (positiv) und die Y-Achse geht aus dem Bildbereich heraus, genau auf den Betrachter vor dem Bildschirm zu. Der Bereich vor der Bildschirmebene entspricht der negativen Y-Achse.

Diese Koordinaten lassen sich mit dem Befehl `(-> Objekt ,absolute-transform)` abrufen, sie werden in Metern angegeben. Im folgenden verwenden wir den Ausdruck „Real-World-Koordinaten“ für sie.

Für eine korrekte 3D-Darstellung im Stereo-Modus müssen die Positionen der Augen des Users angegeben werden. Die Augen bzw. die Nasenwurzel einer Versuchsperson sollen laut Versuchsdesign 0,6 m vom Bildschirm entfernt sein. Die Pupillenmittelpunkte liegen durchschnittlich 3 cm links (-0.03) und rechts (0.03) der Nasenwur-

zel. So ergeben sich folgende Werte, die in der Datei av-stereo-monitor.scm gesetzt werden:

```
(-> (-> left-eye  'Matrix) 'set-value (make-trans-mat -0.03 -0.6 0)) ;;
physical eye dist

(-> (-> right-eye 'Matrix) 'set-value (make-trans-mat  0.03 -0.6 0)) ;;
physical eye dist
```

Folgende Befehle in der Shell eingegeben, stellen die Matrizen der Augenpunkte dar:

```
av-elk> (display (-> av-left-eye 'absolute-transform))
]#[mat
 a11 a12 a13 a14: 1.000000 0.000000 0.000000 0.000000
 a21 a22 a23 a24: 0.000000 1.000000 0.000000 0.000000
 a31 a32 a33 a34: 0.000000 0.000000 1.000000 0.000000
 a41 a42 a43 a44: -0.030000 -0.600000 0.000000 1.000000
```

Wie man in der letzten Reihe der Matrix sehen kann, sind die Real-World-Koordinaten des linken Auges: (-0.03, -0.60, 0.0). Der User sitzt also 60 cm vor dem Bildschirm und sein linkes Auge ist um 3 cm entlang der negativen X-Achse (nach links) verschoben.

Analog treffen diese Aussagen auf das rechte Auge zu, mit dem Unterschied, das die X-Koordinate ein positive Vorzeichen hat:

```
av-elk> (display (-> av-right-eye 'absolute-transform))
#[mat
 a11 a12 a13 a14: 1.000000 0.000000 0.000000 0.000000
 a21 a22 a23 a24: 0.000000 1.000000 0.000000 0.000000
 a31 a32 a33 a34: 0.000000 0.000000 1.000000 0.000000
 a41 a42 a43 a44: 0.030000 -0.600000 0.000000 1.000000
```

Da unsere Tetraeder die Seitenlänge 1 haben, würden sie im Stereo-Modus 1 Meter groß dargestellt werden. Da dies aber viel zu groß ist, sind Target und Cursor noch zu skalieren. Dies geschieht am einfachsten, indem wir einen Knoten (ScaleDCS) anlegen und CursorTrans und TargetTrans, mitsamt ihren Kindern, an ScaleDCS anhängen und ScaleDCS mit dem Faktor 0.06 skalieren.

```
; add the geometry to the root of the scene
(define ScaleDCS (make-instance-by-name "fpDCS"))
(fp-set-value ScaleDCS 'Name "ScaleDCS ")
(fp-set-value ScaleDCS 'Matrix (make-scale-mat 0.06 0.06 0.06))
(fp-set-value ScaleDCS 'Children (list CursorTrans TargetTrans))

(av-add ScaleDCS)
```

Beim Cursor ist die Darstellungsgröße natürlich nicht fest, denn nur so ergibt sich unter anderem der räumliche 3D-Eindruck. Verschiebt man den Cursor nach hinten in den Raum hinein, wird er kleiner, entgegengesetzte Bewegungen lassen ihn größer werden.

Real-World-Koordinaten der Objekte:

Targetpostition: **(0.0,-0.12, 0.00)**

Cursorstartposition 1: **(-0.15, 0.0, 0.09)**

Cursorstartposition 2: **(0.15, 0.0, 0.09)**

Cursorstartposition 3: **(-0.15, 0.0 -0.09)**

Cursorstartposition 4: **(0.15, 0.0 -0.09)**

5.3.7 Toleranz und Distanzvektoren

Die Toleranz für jede Ecke beträgt 0.005 m (5 mm) Wir haben diesen Wert empirisch ermittelt, indem wir während der Programmierphase mit verschiedenen Werten experimentiert haben. Der Wert ist so gewählt, dass man mit einiger Übung recht schnell docken kann. Eine ungeübte Person wird wahrscheinlich am Anfang öfters “durchrutschen”, da er das Gerät noch nicht sicher beherrscht, jedoch kann so auch gut der Lernerfolg mit dem Umgang des Gerätes gemessen werden.

Um herauszufinden, ob eine Ecke des Cursors im Toleranzbereich der zugehörigen Target-Ecke ist, nehmen wir den Betrag des Distanzvektors der beiden Ecken und vergleichen ihn mit der Toleranz. Ist der Betrag kleiner als die Toleranz, so wird ein Flag gesetzt und die Farbe beider Eckmarkierungen ändert sich von rot zu blau. Verlässt die Cursor-Ecke den Toleranzbereich wieder, so werden beide Eckmarkierungen wieder rot.

Realisiert wird die Berechnung mit Hilfe von Distance-vec3, einer Avangofunktion, die von zwei Koordinaten die euklidische Distanz berechnet. Als Parameter benötigt sie zwei Vektoren, welche (get-mat-row (...) 3) liefert. Die Distanzberechnung der Ecken wird mit Real-World-Koordinaten durchgeführt.

```
(define _toleranz 0.005)
...
(set! tmp_tol1 (abs (distance-vec3
(get-mat-row (-> Cursor-ecke-1 'absolute-transform) 3)
(get-mat-row (-> Target-ecke-1 'absolute-transform) 3)
)))
...
(if   (> _toleranz tmp_tol1)
      (begin
            (set! Ecke-1-Flag 1) ;innerhalb der toleranz
      );endbegin
```

```
)

(if (< _toleranz tmp_tol1)
      (begin
            (set! Ecke-1-Flag 0)
      )
);endif
```

5.3.8 WriteOut

Es existieren 2 Arten von WriteOut Files:

- Für ein Trial (LogFileName1-12)
- Für eine komplette Session (LogFileNameAll)

Die Trial WriteOut Files werden eindeutig anhand folgender Daten generiert:

- Personennummer
- Name
- Gerätetyp
- Session
- Trial

Die Session WriteOut-Files werden eindeutig anhand folgender Daten generiert:

- Personennummer
- Name
- Gerätetyp
- Session

Folgenden Daten werden zur weiteren Analyse im Ordner /WriteOutData abgelegt:

- Zeit
- Normierte Rohwerte des Gerätes
- die Translationswerte des Cursors
- Eulervektoren der CursorMatrix (Euler Distanz)
- Toleranzen (Abweichung beim Docking)

Das Herausschreiben wird in 2 Dateien realisiert. Zum einen der functions_and_writeout.scm und zum anderen der callback.scm.

5.3.8.1 Funktionen in functions_and_writeout.scm

Funktionen	Zweck
WriteOutFunctionHeader	für jedes Trial LogFile wird ein Header erstellt, der Informationen zur Versuchsperson, zum Gerät und zum Trial, sowie die erste Zeile der WriteOut - Tabelle enthält. | (senkrechter Strich) dient dabei als Trennzeichen für SPSS.
WriteOutFunction	zuständig für das Rausschreiben der gerundeten (auf 3 Nachkommastellen) Rohdaten des Gerätes sowie der Cursor Rotations- und Translationswerte
WriteOutToleranz	Erstellen eines Hilfslogfiles zum Rausschreiben der Toleranzen
StartPosChoiseTrials	diese Funktion ist nur dann definiert, wenn keine Übung vorliegt (Definition im config.txt), sie schreibt die Toleranzen in das File des vorhergehenden Trials (siehe WriteOutToleranz) und setzt neue Startpositionen für den Cursor
StartPosChoiseUebung	wird nur definiert, wenn Übung vorliegt (Definition im config.txt),. hier entfällt das Rausschreiben der Daten

5.3.8.2 WriteOutFunction:

```
(define (WriteOutFunction LogFileXY Time ValueX ValueY ValueZ ValueRX
ValueRY ValueRZ CursorTrans CursorRot)
```

- LogFileXY - aktuelles Logfile
- Time - Zeit
- ValueX, ValueY, ValueZ - normierte Rohdaten Translation des Gerätes
- ValueRX, ValueRY, ValueRZ - normierte Rohdaten Rotation des Gerätes
- CursorTrans - Translationsmatrix
- CursorRot - Eulerwinkel für Rotation (siehe Callback)

Des weiteren wird zu jedem Frame neben den oberen Werten noch folgende Informationen ausgegeben:

```
(write PersonalNumber   LogFileXY)         (write "|" LogFileXY)
(write TestSubjectName  LogFileXY)         (write "|" LogFileXY)
(write TestSubjectAge   LogFileXY)         (write "|" LogFileXY)
(write DeviceOrder LogFileXY)              (write "|" LogFileXY)
(write DeviceType LogFileXY)               (write "|" LogFileXY)
(write DeviceTypeFlag LogFileXY)           (write "|" LogFileXY)
(write Session LogFileXY)                  (write "|" LogFileXY)
(write TrialCounter LogFileXY)             (write "|" LogFileXY)
```

gerundete Rohdaten (für andere analog)

```
(write (/ (inexact->exact (* ValueX  1000)) 1000) LogFileXY) (write "|"
LogFileXY)
```

Ausgabe der gerundeten Matrixwerte Zeile3, Spalte 0-2

```
(write (/ (inexact->exact (* (mat-ref CursorTrans 3 0) 10000)) 10000)
LogFileXY) (write "|" LogFileXY)
(write (/ (inexact->exact (* (mat-ref CursorTrans 3 1) 10000)) 10000)
LogFileXY) (write "|" LogFileXY)
(write (/ (inexact->exact (* (mat-ref CursorTrans 3 2) 10000)) 10000)
LogFileXY) (write "|" LogFileXY)
```

Ausgabe der Eulervektoren

```
(write (/ (inexact->exact (* (vec3-ref CursorRot    0) 10000)) 10000)
LogFileXY) (write "|" LogFileXY)
(write (/ (inexact->exact (* (vec3-ref CursorRot    1) 10000)) 10000)
LogFileXY) (write "|" LogFileXY)
(write (/ (inexact->exact (* (vec3-ref CursorRot    2) 10000)) 10000)
LogFileXY) (write "|" LogFileXY)
```

5.3.8.3 WriteOutToleranz

Schreiben der Toleranzen der vier Ecken tmp_tol1 – tmp_4

```
(define (WriteOutToleranz LogFileXY-1 tmp_to11 tmp_tol2 tmp_tol3 tmp_tol4)
      (begin
            (newline LogFileXY-1)
            (write "Tmp_Tol 1-4" LogFileXY-1)

            (write tmp_tol1 LogFileXY-1) (write "|" LogFileXY-1)
            (write tmp_tol2 LogFileXY-1) (write "|" LogFileXY-1)
            (write tmp_tol3 LogFileXY-1) (write "|" LogFileXY-1)
            (write tmp_tol4 LogFileXY-1) (write "|" LogFileXY-1)
            (newline LogFileXY-1)
```

Ausgabe folgender Information im Session LogFile für jeden der 12 Trials

```
            (write PersonalNumber         LogFileAll) (write "|" LogFileAll)
            (write TestSubjectName        LogFileAll) (write "|" LogFileAll)
            (write DeviceOrder            LogFileAll) (write "|" LogFileAll)
            (write DeviceType             LogFileAll) (write "|" LogFileAll)
            (write DeviceTypeFlag         LogFileAll) (write "|" LogFileAll)
            (write Session                LogFileAll) (write "|" LogFileAll)
            (write _trialcounter-1        LogFileAll) (write "|" LogFileAll)
            (write _tct                   LogFileAll) (write "|" LogFileAll)
```

```
            (write tmp_tol1              LogFileAll) (write "|" LogFileAll)
            (write tmp_tol2              LogFileAll) (write "|" LogFileAll)
            (write tmp_tol3              LogFileAll) (write "|" LogFileAll)
            (write tmp_tol4              LogFileAll) (write "|" LogFileAll)
            (newline LogFileAll)
      )
)
```

5.3.9 Aufrufe und Implementierung im Callback.scm

Hier befinden sich die Schleifen um einen Durchlauf mehrerer Trials (12) zu gewährleisten. Aus diesem Grund wird hier zweckmäßigerweise das Herausschreiben an diese Schleifen gekoppelt, da für jeden Durchlauf ein File geschrieben wird. Der Callback greift dazu auf die in der functions_and_writeout.scm definierten Funktionen zurück.

5.3.9.1 WriteOutToleranz im Callback

Die Toleranzen werden immer zu Beginn des nächsten Trials in den vorhergehenden geschrieben, deshalb ist es notwendig bei 12 Trials, einen „virtuellen“ 13ten Trial einzuführen.

```
(if (and (< TrialCounter 13)(= TrialFlag 1))   (StartPosChoiseTrials
TrialCounter Session tmp_tol1 tmp_tol2 tmp_tol3 tmp_tol4))
```

Um das Herausschreiben auch bei Trial Nummer 12 zu ermöglichen wird folgende if-Schleife realisiert.

```
(if (and (= TrialCounter 13)(= TrialFlag 1))
(begin
(set! LogFileXY-1 LogFile12)
(WriteOutToleranz LogFileXY-1 tmp_tol1 tmp_tol2 tmp_tol3 tmp_tol4)
(set! DockingFlag 1)
(quit)
)
)
```

Einfügen der aktuellen Zeit in das Log File

```
(if (and (= DockingFlag 0) (= Spacebar 1))
(begin
(set! _time (- _timenew _starttime))
(-> (-> DockingTime 'FloatString) 'set-value (/ (round (* _time 10)) 10))
(if (= TrialCounter 1) (set! _LogFileXY LogFile01))
.
.
.
(if (= TrialCounter 12)(set! _LogFileXY LogFile12))
```

5.3.9.2 WriteOutFunctionHeader im Callback

Ausgabe des Headers ist jeweils nur einmal pro .txt File notwendig.

Aus diesem Grund verknüpft man den Header WriteOut mit einer Markierung, einem sog. Flag.

```
(if (= WriteOutFlag 1)(WriteOutFunctionHeader _LogFileXY))
```

Ist das WriteOutFlag auf 1 gesetzt so wird die WriteOutFunction ausgeführt und der Header wird geschrieben. Innerhalb der Funktion wird der Flag wieder auf 0 gesetzt. Somit wird gewährleistet das nur einmal pro File, der Header geschrieben wird. Ist ein Trial beendet, also das Docking war erfolgreich, dann wird der WriteOutFlag wieder auf 1 gesetzt.

5.3.9.3 WriteOutFunction Callback

Bestimmung der Eulerwinkel aus der Rotationsmatrix

```
(set! _CursorRotMatrix (get-mat-euler (fp-get-value CursorRot   'Matrix)
```

Der WriteOut der Rohdaten sowie der Cursorpositionen geschieht in Abhängigkeit vom ausgewählten Gerät in der config.txt. Da die Geräte Unterschiede in der Sensorik aufweisen, ist es notwendig jedes Gerät gesondert zu betrachten, da die Rotation bei der SpaceMouse z.B. anders realisiert wird, als das bei der Kugelmaus der Fall ist.

SpaceMouse:

```
(if (equal? DeviceTypeFlag "a");Spacemouse
(WriteOutFunction _LogFileXY _time _valueX _valueY _valueZ _valueRX
_valueRY _valueRZ _CursorTransMatrix _CursorRotMatrix))
```

grosser Kugelfisch:

```
(if (equal? DeviceTypeFlag "b");grKF
(WriteOutFunction _LogFileXY _time _valueX _valueY _valueZ KugelRX KugelRY
KugelRZ _CursorTransMatrix _CursorRotMatrix))
```

Kugelmaus:

```
(if (equal? DeviceTypeFlag "c" );Kugelmaus
(WriteOutFunction _LogFileXY _time _valueX _valueY _valueZ KugelRX KugelRY
KugelRZ _CursorTransMatrix _CursorRotMatrix))
```

kleiner Kugelfisch:

```
(if (equal? DeviceTypeFlag "d");klKF
```

```
 (WriteOutFunction _LogFileXY _time _valueX _valueY _valueZ KugelRX KugelRY
KugelRZ _CursorTransMatrix _CursorRotMatrix))
```

5.3.10 Hinweis zu Verzeichnisstruktur

Unser Programm sowie die Rohdaten liegen zur Zeit auf dem Xchange-Verzeichnis des VR-Labs unter: /home/avango/xchange/EvaGrip2-Final>

Folgende Verzeichnisse und Dateien gehören zu unserem Experiment:

Verzeichnis Inhalt/Zweck	**enthaltene Dateien**	
WriteOutData in diesen Ordner werden die Dateien vom Programm geschrieben.	jeweils die aktuellen 48 + 4 Files pro Gerät	
geometry Inventor Files der verwendeten Geometrien	sphere_blau.iv sphere_rot.iv star_slim_blue.iv star_slim_red.iv tetraeder_scaled_colored_slim_centerend.iv tetraeder_scaled_colored_slim_centered_2.iv	
(material)*	av-stereo-monitor.scm stripes_blue_144_1.rgb stripes_blue_15.rgb bodenplatte.jpg stripes_blue_15.gif stripes_slim_blue_144.rgb	
scripts (scripts/uebung)* Scheme-Dateien	av-linux-event.scm callback.scm config.txt device_grK.scm display.scm evagrip2-zahi-replikation-tetraeder.scm (eyepoint.scm)* functions_and_writeout.scm geometry.scm mouse-ptr.scm (writeout.scm)*	devices.scm devices_KM.scm devices_SM.scm devices_grK.scm devices_klK.scm grKugelfisch.scm klKugelfisch.scm (klKugelfisch_old.scm)* kugelmouse.scm SpaceMouse.scm
dataVP	Die Daten von 16 Versuchspersonen, pro VP ein Ordner.	

*: Anmerkung: Die eingeklammerten Dateien und Verzeichnisse werden für den Task nicht mehr gebraucht. Sie sind noch vorhanden um ggf. Modifikationen vornehmen zu können.

5.4 Quaternionen

5.4.1 Grundlagen Mapping

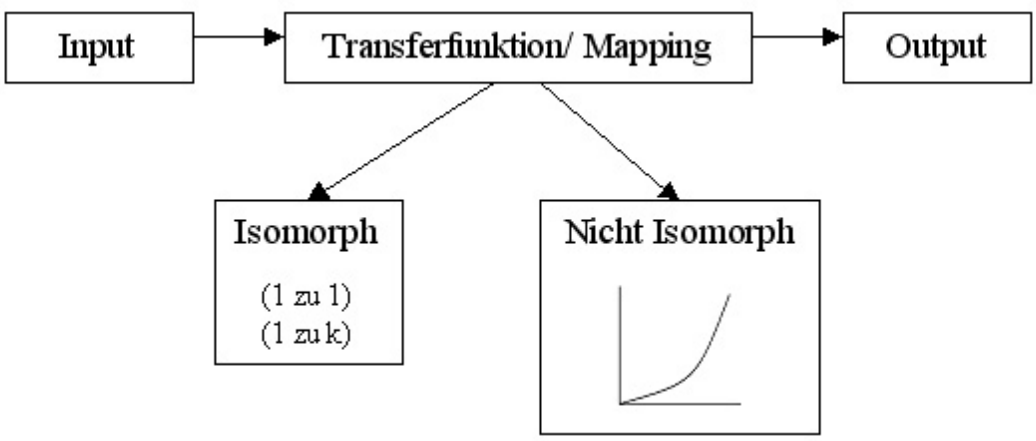

Abb. 5.3: Mapping-Typen

Nicht-isomorphe Mappings sind bei Translation schon lange im Einsatz und auch sehr erfolgreich. Für Rotationen finden dagegen weitgehend nur isomorphe Verwendung. Die Effizienz der isomorphen Rotation wird jedoch durch die Mechanik/Technik der Eingabegeräte und den menschlichen Körperbau beschränkt. Beispiel für Ersteres sind die Aufhängungen der Kugelgeräte. Damit die Kugel fixiert ist, muss eine Bügel-/ Klammervorrichtung Teile der Kugel umschließen.

Der Gelenk- und Muskelapperat des Menschen unterliegt ebenfalls physikalischen Beschränkungen. Um die Kugel einmal komplett zu drehen muss mehrmals Losgelassen und Nachgefasst werden. Dieser Umgreifprozess wird reclutching genannt und kostet natürlich Zeit.

Nicht-isomorphe Techniken versprechen hier effektiveres Arbeiten, da die Häufigkeit der Reclutches reduziert werden kann.

Die Studie „Non-Isomorphic 3D Rotational Techniques“ von Ivan Poupyrev, Suzanne Weghorst, Sidney Fels befasst sich mit der Effizienz von isomorphen und nicht-isomorphen Rotationstechniken. Die Ergebnisse zeigen, dass mit nicht-isomorphen

Rotationsmapping bei großen Rotationsvorgängen schneller und einfacher gearbeitet werden konnte.

Die Umsetzung von nicht-isomorphen Rotationen ist jedoch nicht trivial.

Rotationen können am einfachsten als Teilrotationen um die einzelnen Achsen verstanden werden.

Beispiel: Eine Kugel soll schräg rotiert werden (Abb. 5.3). Die Rotation besteht aus einer kleineren X-Komponente und einer größeren Y-Komponente. Sie wird also als Kombination aus Rotationsanteilen um zwei Achsen angesehen.

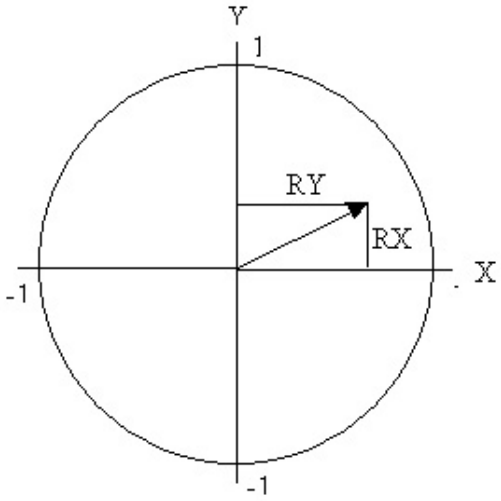

Abb. 5.4: zur Vereinfachung Kugel hier nur planar dargestellt; Aufrissansicht der Kugel verwendet.

Für nicht-isomorphe Mappings werden üblicherweise Exponentialfunktionen als Transferfunktion eingesetzt. Beide Rotationsanteile müssten also expotenziert werden. Die Beträge der Rotationskomponenten würden dann unterschiedlich stark verstärkt werden. Die Rotationsrichtung bleibt nicht konstant, sondern verschiebt sich zu Gunsten der stärkeren Rotationskomponente. Die Rotationsrichtung wird verfälscht (Abb. 5.5).

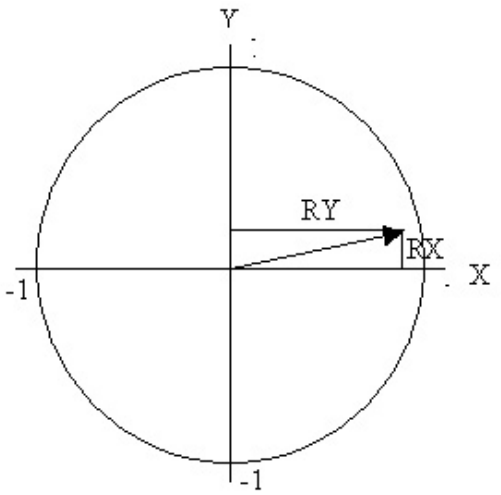

Abb.5.5: Teilkomponenten unterschiedlich stark verstärkt

Im Verlauf mehrerer Messzyklen wird die Abweichung/ der Fehler immer größer. Die gemappte Rotationsrichtung würde sich in Form einer Kurve immer weiter von der tatsächlichen Rotationsrichtung entfernen (Abb. 5.6).

Rotationsbeschreibung durch Teilkomponenten ist hier also nicht sinnvoll. Eine Lösung sind Quaternionen.

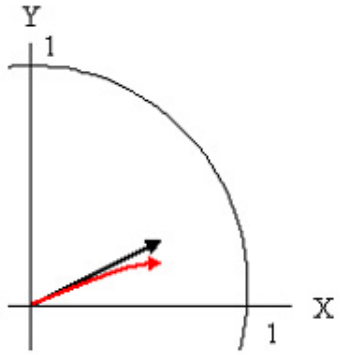

Abb. 5.6: Tatsächlicher Weg vs. gemappter Weg

5.4.2 Grundlagen Quaternionen

Quaternionen stellen ein alternatives Modell zur Beschreibung von Rotationen dar. Ein Quaternion ist ein Tupel (s,x,y,z), wobei s ein Skalar und (x,y,z) einen Vektor darstellt. Die geometrische Beschreibung ist einfach: Der Vektor (x,y,z) definiert ein Achse im 3D Raum und s ist der Winkel, mit dem um diese Achse gedreht wird. Beispiel: Eine Kugel wird beliebig gedreht (Abb. 5.7).

Der blaue Kreis steht für die Drehrichtung der Rotationsbewegung. Ein markanter Punkt Rot wird während der Drehung verfolgt. Er „wandert“ von rechts oben nach links unten. Mit einem Quaternion wird diese Drehung als ein einziger Vorgang angesehen. Die Drehrichtung wird durch die Rotationsachse (x,y,z) spezifiziert, die Stärke der Drehung durch den Winkel ά. (Abb. 5.8) Ein Quaternion wird nach folgender Formel berechnet. u ist hier der 3D-Vektor der Rotationsachse:

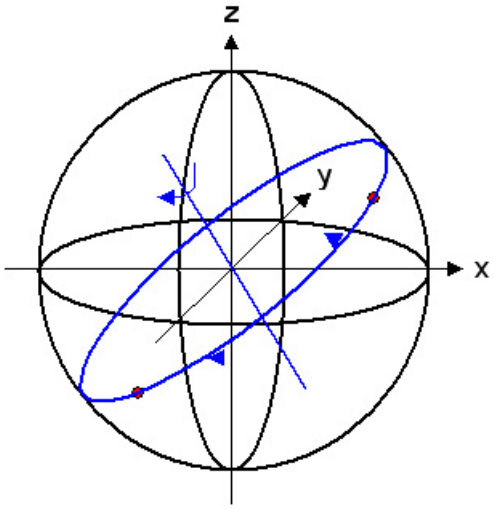

Abb.5.7

$$q = (\sin\frac{\alpha}{2}\mathbf{u}, \cos\frac{\alpha}{2})$$

Soll eine Rotation in Quaternionendarstellung verstärkt werden, wird einfach ein Verstärkungsfaktor mit dem Winkel verrechnet. Die Rotationsachse bleibt unberührt. Lediglich der Betrag des Winkels ändert sich. Die Formel für ein beschleunigtes Quaternion:

$$q = (\sin\frac{k\alpha}{2}\mathbf{u}, \cos\frac{k\alpha}{2})$$

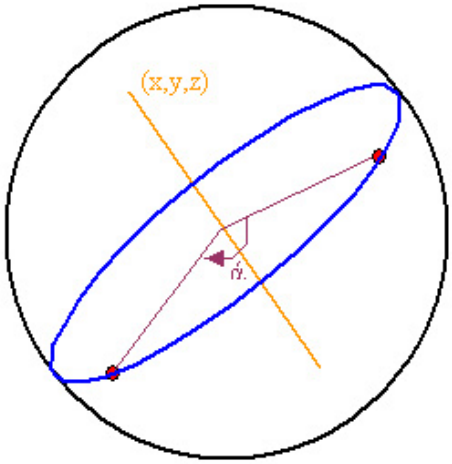

Abb. 5.8

Quaternion kompakt:

- korrekte Rotationsbeschreibung
- keine Reihenfolgeeffekte zu beachten
    - kein Gimbal Lock (Verlust eines DoF)
- Rotationen einfach zu verstärken
- Interpolationen zwischen zwei Quaternionen einfach zu realisieren.
  Anwendung bei Animation von einer Objektorientierung in eine andere

5.4.3 Geometrische Erklärung unseres Messverfahrens:

2 optische Sensoren S1 und S2 messen jeweils 2 Dimensionen

Deren Werte werden als Richtungsvektoren interpretiert und entsprechen den Rotationsanteilen um die einzelnen Achsen (Abb. 5.9).

Sensor1 misst Rotation um X (RX) und RZ, während Sensor2 RY und erneut RZ misst. RZ ist also überbestimmt. Eine RZ-Messung wird verworfen.

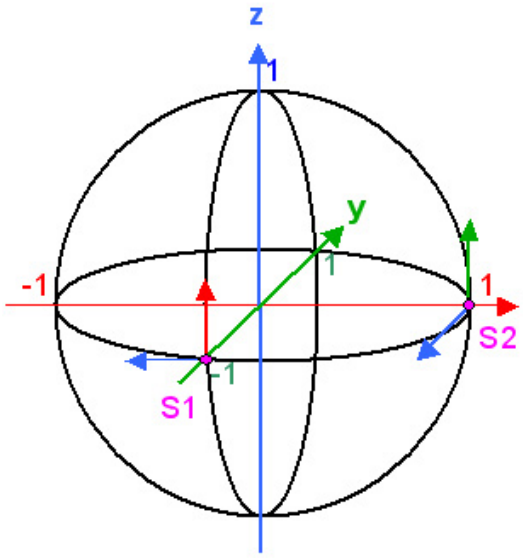

Abb. 5.9

Mann kann sich ja ein Punkt vorstellen, der zu Beginn jedes Frames im Nullpunkt des planaren Koordinatensystems eines jeden Sensors liegt. Dieser Punkt wird durch die gemessenen Inputs dieses Frames einmalig verschoben. Die Nullstellung des Punktes entspricht z.B. S1, der Punkt nach der Verschiebung S1´ (Abb. 5.10). Bei zwei Sensoren haben wir auch zwei wandernde Punkte S1 und S2. Im nächsten Frame wird der Punkt wieder in den Ursprung gesetzt und der Prozess beginnt von neuem.

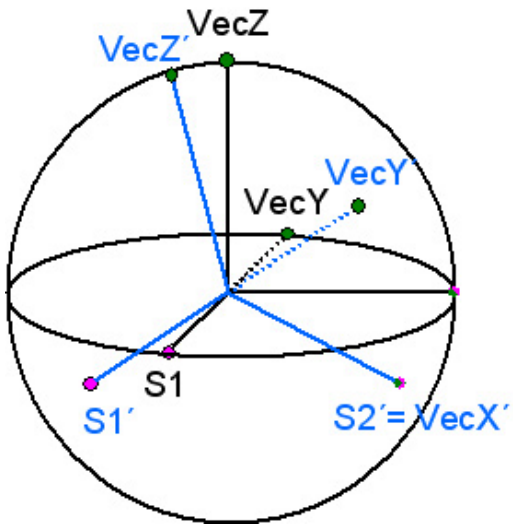

Abb. 5.10

Weiterhin können von einer Einheitskugel ausgehen. Doch diesmal möchten wir nicht im 2D Koordinatensystem jedes Sensors arbeiten sondern direkt im 3D Einheitskoordinatensystem der Kugel. Wir definieren daher Punkt VecX (1|0|0), welcher vorher Punkt S2 war, und Punkt VecY (0|1|0), der gegenüberliegende Punkt von S1, als Richtungsvektoren.

Zum Aufstellen einer Rotationsmatrix benötigen wir jedoch 3 bekannte Punkte. Wir definieren daher zusätzlich Punkt VecZ, zunächst mit (0|0|1) als Nordpol der Kugel (Abb. 5.11). Die Verschiebungen von VecX und VecY erhalten wir direkt aus S1 und S2, VecZ bleibt unbeeinflusst von den Sensorenwerten. Da aber VecZ sowohl zu VecX als auch zu VecY orthogonal ist, kann VecZ durch das Kreuzprodukt berechnet werden. Die Anforderung der Einheitskugel muss weiterhin gelten, daher werden die Vektoren normalisiert.

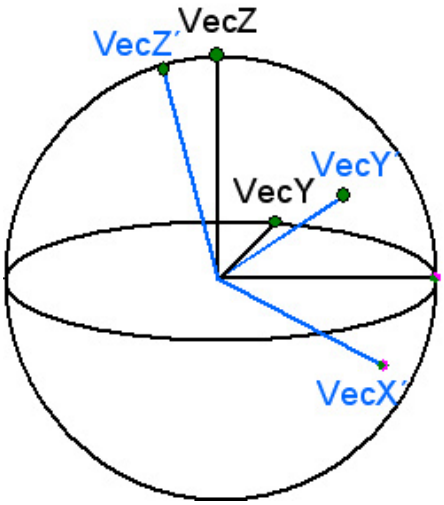

Abb. 5.11

Eine Rotationsmatrix kann aufgestellt werden und wird anschließend invertiert. Nun können die VecPunkte wieder auf ihre Nullstellung gesetzt werden, um im nächsten Frame wieder Veränderungen zu messen.

Aus der invertierten Rotationsmatrix wird ein Quaternion erstellt. Scheme stellt diese Funktionalität zur Verfügung. Aus dem Quaternion wird der Winkel entnommen. Er ist nicht direkt ersichtlich, sondern muss erst zurückgerechnet werden.

$$q = (\sin\frac{\alpha}{2}\mathbf{u}, \cos\frac{\alpha}{2})$$

Ein Quaternion kann durch einen Faktor k vor dem Winkel verstärkt werden.

$$q = (\sin\frac{k\alpha}{2}\mathbf{u}, \cos\frac{k\alpha}{2})$$

In unserem Fall wollen wir keinen linearen Verstärkungsfaktor, sondern eine Nicht Isomorphe Verstärkung verwenden. Dies erscheint sinnvoll, da durch die Aufhängung der Kugel der tatsächliche Freilauf beschränkt wird. Von Rahmen zu Rahmen stehen ohne Umgreifen ca. 90° der Kugeloberfläche zur Verfügung. Dieser Wert könnte durch einen Exponenten erhöht werden.

Eine Eigenschaft von Exponentialfunktionen (Abb. 5.12) besteht darin, dass sie bei Werten <1 einen kleineren Rückgabewert aufweisen als der Eingangswert. Hier würde die Rotation verlangsamt. Die sehr präzisen Rotationseinstellungen zum Feindocking, werden so leichter ausführbar. Wir hätten diesen Verlangsamungseffekt auch gerne für größere Quaternionenwinkel. In Testversuchen

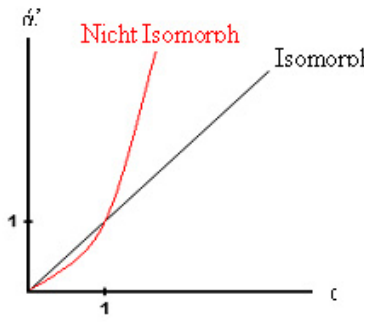

Abb. 5.12

haben wir 35° für einen geeigneten Wert empfunden. Man dividiert also durch 35 und „simuliert“ so den Bereich <1. Der Exponent wird ausgeführt, die Rotation verlangsamt. Danach muss mit 35 multipliziert werden, um wieder in den ursprünglichen Bereich zu kommen. Die Abbildung 5.13 zeigt skizzenhaft den Verlauf unserer Verstärkungsfunktion.

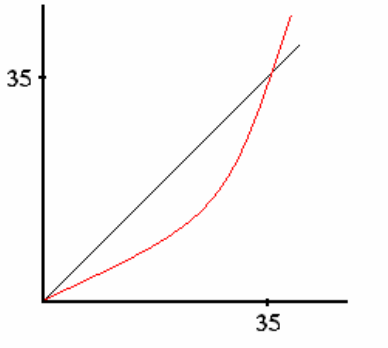

Abb. 5.13

Kleine Zusammenfassung: Quaternionenwinkel <35° werden verkleinert, um das Feindocking (am Ende der Trials) zu erleichtern. Winkel >35 werden vergrößert, um große Orientierungsdiskrepanzen (zu Beginn der Trials) schnell und mit wenig Umgreifen zu überwinden.

5.4.4 Erläuterung der Quaternionenimplementation anhand des kleinen Kugelfischs

Zunächst werden die beiden optischen Sensoren initialisiert

```
(define fish-sensor2 (make-instance-by-name "fpADSensor"))
(fp-set-value fish-sensor2 'Station  "usb-station2")

(define fish-sensor3 (make-instance-by-name "fpADSensor"))
      (fp-set-value fish-sensor3 'Station  "usb-station3")
```

... und anschließend sogleich erste Werte abgenommen. Jeder der beiden Sensoren misst 2 Dimensionen. Man erhält also 4 Werte. Für Rotationen werden natürlich nur 3 Werte (3 Winkel) benötigt. Ein Winkel wird durch die Anordnung der Sensoren zueinander von beiden Sensoren gemessen, bei uns ist dies der Z-Winkel. Durch Testen der Ausschläge auf diesen beiden Z-Kanälen stellte sich heraus, dass beide Sensoren exakt den gleichen Wert für diesen Winkel zurückgeben. Man kann demzufolge von Redundanz sprechen. Der zweite erhobene Z-Wert wird verworfen. Die Referenzvektoren VecX, VecY, VecZ und ein Verstärkungsfaktor werden initialisiert…

```
(define _f2x_old   (* (fp-get-value fish-sensor2 'Value0) -1)) ; Rohwert
Rot x
(define _f2y_old   (* (fp-get-value fish-sensor2 'Value1) -1)) ; Rohwert
Rot z
```

```
(define _f3y_old   (* (fp-get-value fish-sensor3 'Value0) 1))  ; Rohwert
Rot y
(define _f3x_old   (* (fp-get-value fish-sensor3 'Value1) 1)) ; Rohwert Rot
z

(define VecX (make-vec3 1.0 0 0))
(define VecY (make-vec3 0 1.0 0))
(define VecZ (make-vec3 0 0 0))
(define k 1.3) ;; Verstärkungsfaktor für Quaternionenrotation
```

... Damit ist der statische Teil beendet. Ab jetzt beginnt der Callback, also der dynamische Teil dieser Anwendung. Der Callback wird vom Zeitsensor ausgelöst. Jedes Durchlaufen des Callbackinhaltes stellt einen Frame dar. Es werden sofort wieder die Werte der 3 Inputkanäle gemessen…

```
; Beginn des Callback
(define (fetchCB value)
      (let (

            (_f2x   (* (fp-get-value fish-sensor2 'Value0) -1)) ; z
            (_f2y   (* (fp-get-value fish-sensor3 'Value0) 1))  ; z
            (_f3y   (* (fp-get-value fish-sensor2 'Value1) 1)) ; y
            (_f3x   (* (fp-get-value fish-sensor3 'Value1) 1)) ; x
)
```

... Anschließend wird für jeden Kanal der Wert zu Frame N vom Wert Frame N-1 abgezogen. So erhält man einen Differenzialwert der die Veränderung vom vorhergehenden zum aktuellen Frame darstellt. Der gewonnen Wert wird in der neuen Variable KugelRX, -RY, -RZ gespeichert.

Der aktuelle Wert auf dem Kanal wird in der entsprechenden _old Variable gespeichert, um einen Referenzwert für den nachfolgenden Frame zu erhalten, damit dann wieder die aktuelle Veränderung ausgerechnet werden kann, usw. …

```
;; (set! KugelRX (- _f3y _f3y_old))

                  (set! KugelRX (- _f3x _f3x_old))
                  (set! KugelRY (- _f3y _f3y_old))
                  (set! KugelRZ (- _f2y _f2y_old))

                  (set! _f2x_old _f2x)
                  (set! _f3y_old _f3y)
                  (set! _f2y_old _f2y)
                  (set! _f3x_old _f3x)
```

... Die Beträge der Sensoren sind jeweils ganzzahlige Werte. Die Auflösung sind Einser-Schritte. Der gewonnene Differenzialwert ist demzufolge auch ganzzahlig. Er weist aber leider keinen erkennbaren Maximalwert auf. Wenn man sehr schnell an der Kugel dreht, erhält man Werte von >200. Tests zeigten, dass eine realistischer Maximalwert +/-90 ist. Dabei wird die Kugel mit 90°/sec gedreht. Alles, was vom

Betrag größer als 90 ist, wird genau auf diesen Maximalwert 90 reduziert. Anschließend wird eine Normalisierung auf den Bereich [-1; 1] vorgenommen. …

```
(if (> KugelRX 90) (set! KugelRX 90))
                  (if (< KugelRX -90) (set! KugelRX -90))
            (if (> KugelRY 90) (set! KugelRY 90))
                  (if (< KugelRY -90) (set! KugelRY -90))
            (if (> KugelRZ 90) (set! KugelRZ 90))
                  (if (< KugelRZ -90) (set! KugelRZ -90))

            (set! KugelRX (/ KugelRX 90))
            (set! KugelRY (/ KugelRY 90))
            (set! KugelRZ (/ KugelRZ 90))
```

… Die aktuellen Sensorwerte werden auf die beiden Referenzpunkte der zwei Sensorenfelder (Nullpunkt) übertragen (siehe auch geometrische Beschreibung des Verfahrens Kapitel 5.4.3). Beispiel VecX: Hier bleibt die X-Komponente von VecX unverändert (auf 1).

Die Veränderungen finden lediglich im planaren Unterkoordinatensystem (Sensorenkoordinatensystem) von VecX statt. Die Y-Komponente von VecX steht für die Drehung um die Z-Achse (blaue Pfeile in 5.6.3). Die Z-Komponente von VecX entsteht aus der Drehung um die Y-Achse (grüner Pfeil in 5.6.3).

Beispiel VecY: Die X-Komponente von VecY steht für die Drehung um die Z-Achse (blaue Pfeile in 5.6.3). Sie muss lediglich im Vorzeichen vertauscht werden, da der Sensorenmesspunkt und unser VecY-Referenzpunkt genau gegenüber zueinander auf der Kugel liegen. Zur Erinnerung: Sämtliche Aktionen auf einer Kugel erscheinen auf der gegenüberliegenden Seite richtungsverkehrt. Die Y-Komponente von VecY bleibt unverändert (auf 1). Die Z-Komponente von VecY entsteht aus der Drehung um die X-Achse (roter Pfeil in 5.6.3). Das Vorzeichen wir hier ebenfalls verändert. …

```
(set-all-vec3! VecX (vec3-ref VecX 0)
  (+ (vec3-ref VecX 1) KugelRZ)
  (+ (vec3-ref VecX 2) KugelRY))

(set-all-vec3! VecY (+ (vec3-ref VecY 0) (* KugelRZ -1))
  (vec3-ref VecY 1)
  (+ (vec3-ref VecY 2) (* KugelRX -1)))
```

… Wie beschrieben haben beide Sensoren nur eine 2D (planare) Auflösung. Da sie aber eine Kugeloberfläche abtasten, kann die gemessene Rotation geringfügig von der tatsächlichen Kugeldrehung abweichen. Grund dafür ist die Krümmung der Kugel. Am Rand des Auflösungsfelds der Sensoren, also bei sehr großen Rotationen, ist die Diskrepanz am Größten. In der endgültigen Studienversion der

Geräte wurden daher die zuerst eingesetzten Maussensoren durch Trackballsensoren (Logitech) ersetzt, die speziell dafür ausgelegt sind Kugeloberflächen abzutasten.

Durch Normalisierung der Punkte VecX und VecY stellen wir sicher, dass wir uns immer noch in einer Einheitssphäre befinden. …

```
            (set! VecX (normalize-vec3 VecX))
            (set! VecY (normalize-vec3 VecY))
```

… Der dritte benötigte Punkt kann durch das Kreuzprodukt aus VecX und VecY gewonnen werden. Wieder Normalisierung zur Wahrung der Einheitssphäre.

Wir haben jetzt drei Punkte, die ein Koordinatensystem aufspannen sollen. Dazu müssen jedoch alle Vektoren untereinander orthogonal sein. Dies ist auch fast der Fall. Lediglich die Lage von VecX und VecY zueinander sollte überprüft werden. Zu Beginn des Messprozesses wurde von Orthogonalität dieser Vektoren ausgegangen (siehe VecX (1|0|0) und VecY(0|1|0)). Durch Rundungsfehler oder Abweichung der tatsächlichen Sensorenposition (Sensoren evtl. nicht aufs Grad genau in ihrer geplanten Position verankert) kann die diese Orthogonalität verloren gegangen sein. Man kann jetzt quasi einen Wert „opfern“, wir nehmen VecY, und mittels Kreuzprodukt aus den anderen beiden neu berechnen. Anschließend wieder Normalisieren. …

```
            (set! VecZ (cross-vec3 VecX VecY))
            (set! VecZ (normalize-vec3 VecZ))

            (set! VecY (cross-vec3 VecZ VecX))
            (set! VecY (normalize-vec3 VecY))
```

… Jetzt kann eine Rotationsmatrix aus den X-, Y-, Z-Anteilen der Referenzpunkte VecX, VecY und VecZ erstellt werden. Die gewonnene Rotationsmatrix wird invertiert und entspricht nun der Orientierungsänderung des Objekts im aktuellen Frame. VecV, VecY und VecZ werden nun noch für den nachfolgenden Frame wieder zurück in ihre Ausgangswerte gesetzt. …

```
(define RotMat (make-mat
(vec3-ref VecX 0) (vec3-ref VecX 1) (vec3-ref VecX 2) 0
            (vec3-ref VecY 0) (vec3-ref VecY 1) (vec3-ref VecY 2) 0
            (vec3-ref VecZ 0) (vec3-ref VecZ 1) (vec3-ref VecZ 2) 0
                    0                 0                 0         0))

(set! RotMat (invert-orthon-mat RotMat))

            (set-all-vec3! VecX 1 0 0)
            (set-all-vec3! VecY 0 1 0)
```

```
            (set-all-vec3! VecZ 0 0 1)
```

… Aus Rotationsmatrix wird ein Quaternion berechnet. Der Winkel wird aus dem Quaternion entnommen. Dazu wird die 4. Stelle (das Skalar) des Quaternions genommen und die Umkehrung der Formel angewandt. Ein Faktor realisiert die Umrechnung ins Gradmaß. …

```
(define Quat (get-ortho-mat-quat RotMat)) ;; Quaternion aus Rotationsmatrix
generieren

(define QuatWinkel (* (* (acos (quat-ref Quat 3)) 57.2958) 2)) ;;
Rotationswinkel des Quaternions (Umkehrung der Formel)
```

… Der Quaternionenwinkel wird, je nach seiner Ausgangsgröße, verstärkt oder reduziert und ein neues beschleunigtes Quaternion erstellt. Die Rotationsachse bleibt unverändert. …

```
(if (= (quat-ref Quat 3) 1)
      (begin (define QuatBeschl (make-quat 0 0 0 1))) ;; Sonderfall: nan >>
Quaternion nicht definiert

      (begin (define QuatBeschl (make-quat
(* (expt (/ QuatWinkel 35) k) 35) ; Winkel
(/ (quat-ref Quat 0) QuatWinkel)  ; Achsenkomponenten
(/ (quat-ref Quat 1) QuatWinkel)
(/ (quat-ref Quat 2) QuatWinkel))))
            )
```

… Beschleunigtes Quaternion wird in eine Rotationsmatrix zurück überführt. …

```
(set! RotMat (make-quat-mat QuatBeschl)) ;; beschleunigte Rotationsmatrix
aus Quaternion generieren
```

… Wie in der Gerätevorstellung beschrieben, sitzen die optischen Sensoren nicht genau entlang des Gerätekoordinatensystems, sondern sind aus baulichen Gründen versetzt worden. Bei der KM sind sie zum Beispiel um 45° in Z-Richtung verdreht. Die gewonnene Rotationsmatrix entspricht demzufolge auch dem verdrehten Sensorenkoordinatensystem und muss quasi wieder richtig gedreht werden. Dazu wird das eigentliche Geräte- /Cursorkoordinatensystem (CursorRot) auf Stellung des Sensorenkoordinatensystems gedreht. Beide Systeme stimmen jetzt überein und die gewonnene Rotationsmatrix, die aus den Sensorendaten hervorgeht, kann auf die gedrehte Cursorrotationsmatrix aufmultipliziert werden. Anschließend wird diese Cursorrotationsmatrix wieder auf ihre richtige Stellung zurückgedreht. …

```
(fp-set-value CursorRot 'Matrix (mult-mat
      (fp-get-value CursorRot 'Matrix) (make-rot-mat 45 0 0 1)))
(fp-set-value CursorRot 'Matrix (mult-mat
      (fp-get-value CursorRot 'Matrix) RotMat))
(fp-set-value CursorRot 'Matrix (mult-mat
      (fp-get-value CursorRot 'Matrix) (make-rot-mat -45 0 0 1)))
```

… Die Szenengeometrie (Cursortetraeder) wird entsprechend der Cursorrotationsmatrix gezeichnet. Fertig!

6. Versuchsauswertung

6.1 Auswertung der Task Completion Time

6.1.1 Einleitung und Begriffsklärung

Task Completion Time (TCT): Zeit die bis zum erfüllen der Aufgabe benötigt wurde.

Varianzanalyse (ANOVA: Analysis of Variances): ist eine statistische Analysemethode, um Haupteffekte und Interaktionseffekte zwischen unabhängigen Variablen bzgl. einer abhängigen Variablen aufzudecken.
Bei unserer Varianzanalyse waren die Within-Faktoren:

- Gerät: 3 Ausprägungen (kleiner Kugelfisch, SpaceMouse, Kugelmaus)
- Sessions: 4 Ausprägungen

Und der Between Faktor, war die Gerätereihenfolge.

Signifikanz: Unterschiede heißen signifikant, wenn sie mit einer bestimmten Wahrscheinlichkeit nicht durch Zufall zustande gekommen sind. Die Überprüfung der statistischen Signifikanz geschieht mit Hilfe einer Nullhypothese, die verworfen wird, wenn das zufällige Zustandekommen des Unterschiedes sehr unwahrscheinlich ist. Der Grad der zu überprüfenden Unwahrscheinlichkeit wird vorher festgelegt und mit p bezeichnet. In unserem Fall ist ein Wert signifikant für $p < 0.05$.

Standardabweichung (SA): Ist ein Maß für die Streuung um den Mittelwert. Sie wird auch als mittlerer Fehler bezeichnet.

Lernkurven der Mittelwerte: Für jede Session und jedes Gerät wurden über alle 16 Versuchspersonen die Mittelwerte der TCTs sowie deren SA bestimmt. Diese ergeben grafisch abgetragen die Lernkurven.

6.1.2 Lernkurven für MW und SA

Bei Betrachtung der Lernkurven der TCT und SA fielen einige Unregelmäßigkeiten auf (Abb. 6.1). Zum Beispiel unterschied sich der Verlauf der Lernkurve des kleinen Kugelfischs, der Kugelmaus und der SpaceMouse wesentlich von der des großen

Kugelfischs. Bei der genaueren Betrachtung der Trial-TCT der einzelnen Versuchspersonen stellte sich heraus das Versuchsperson 1 doppelte so hohe TCTs mit allen Geräten erzielte, so dass diese Werte nicht in die endgültige Betrachtungen einbezogen wurden (Abb. 6.2). Dadurch verbesserten sich die TCT Mittelwerte über die 4 Sessions für die vier Geräte wie folgt: Für den großen Kugelfisch von 19,34 s auf 17,03 s für die SpaceMouse von 14,70 s auf 13,84 s für die Kugelmaus von 11,48 s auf 10,85 s und für den kleinen Kugelfisch von 10,09 s auf 9,60 s.

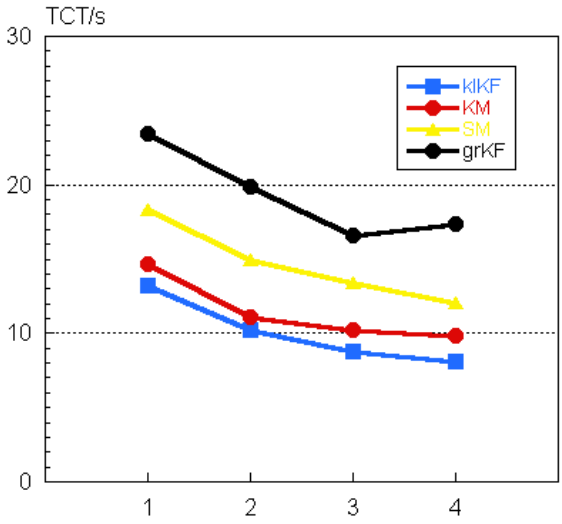

Abb.: 6.1: MW-TCT mit VP01.

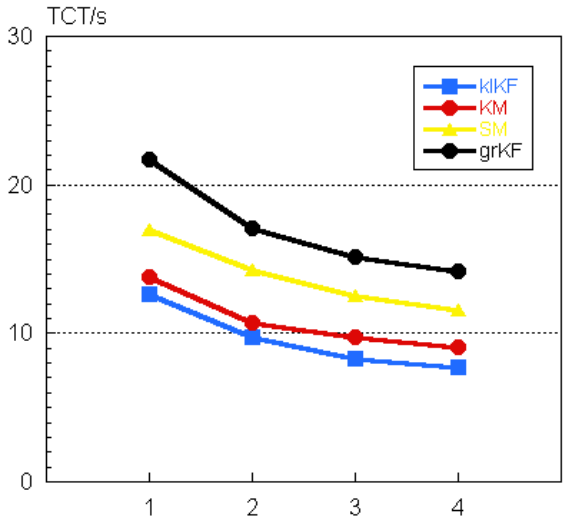

Abb.: 6.2: MW-TCT ohne VP01.

	1	**2**	**3**	**4**	MW
grKF	21,72	17,04	15,19	14,16	17,03
SM	17,02	14,32	12,50	11,54	13,84
KM	13,82	10,74	9,74	9,11	10,85
klKF	12,65	9,79	8,26	7,71	9,60

Mittelwerte der TCT über die 4 Sessions und alle Personen, minus 0.8s, ohne VP1.

Wie man den oberen Messwerten entnehmen kann, lagen die Mittelwerte der Gesamtzeiten bei dem großen Kugelfisch weit über dem (7,43 sec) des von der Bauart ähnlichen kleinen Kugelfischs.

Dies ist zu einem Großteil auf die technische Unausgereiftheit und Fehlfunktionen des großen Kugelfischs zurückzuführen, der somit weit hinter den anderen Geräten zurückfiel (3,19 s gegenüber SpaceMouse) und somit keine vergleichbaren Aussagen zuließ. Dies wurde auch durch die Aussagen der Versuchspersonen in den Fragebögen bestätigt (siehe Fragebogenauswertung). Aus diesen Gründen wurde der große Kugelfisch bei der Datenanalyse nicht berücksichtigt.

6.1.3 Signifikanzen

Die Betrachtung der Gerätereihenfolge ergaben sich keine signifikanten Unterschiede ($F_{3,11}$=1,76, p=0,21). Weiterhin traten keine signifikanten Interaktionseffekte mit den Within Faktoren auf.

- Gerät: $F_{6,22}$=2,13, p=0,12
- Session: $F_{9,33}$<1
- Geräte & Session: $F_{18,66}$=1,51, p=0,20

Die Performance der Geräte unterschied sich signifikant ($F_{2,22}$=24,86, p<0,001).

Mit dem kleinen Kugelfisch benötigte man im Mittel 10,25s bei einer SA von 0,48 um den Task abzuschließen. Mit der Kugelmaus brauchte man 11,47s (SA = 0,53) Die SpaceMouse erzeugte mit 14,62 s im Mittel (SA = 0,84) die höchsten TCTs.

In der Gesamtheit kann man jedoch erkennen das sich die Performance der Geräte bei der Erfüllung des DockingTasks von Session 1 zu Session 4 (F3,33=58,83, p<0,001)verbessert. Brauchten die Versuchspersonen in Session 1 noch 15,11s (SA = 0,81), so betrug sie in Session 2 nur 12,32s (SA = 0,54), 10,86s (SA = 0,47) in Session 3, bis hin zu 10,17s (SA = 0,39) in der vierten Session Diese Steigerung der Performance unterschied sich aber nicht signifikant von Gerät zu Gerät ($F_{6,66}$<1), das bedeutet das der Anstieg der Performance für alle Geräte im gleichen Maße erfolgte.

Im folgenden werden TCT und SA grafisch dargestellt um Lernkurven zu verdeutlichen und Rückschlüsse auf die Haupthypothese zu ziehen.

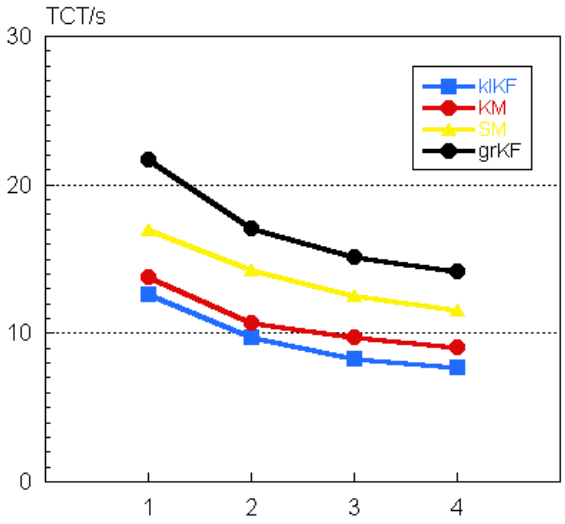

Abb 6.3: MW-TCT ohne VP01

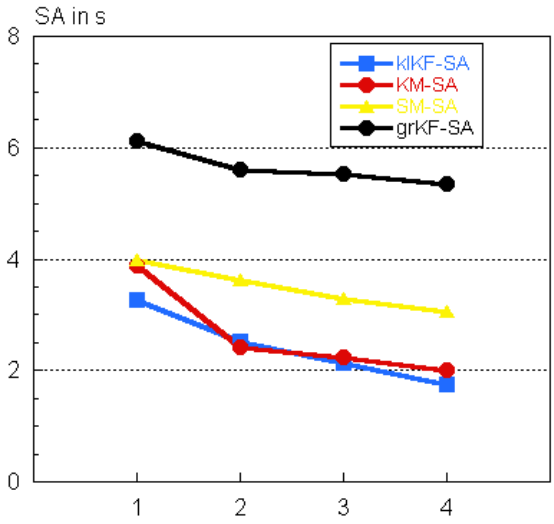

Abb 6.4: MW-SA ohne VP01

In Abbildung 6.3 ist zu erkennen, dass sich, wie oben beschrieben die TCTs von Session 1 zu 4, stetig verbessern. Die besten TCTs wurden mit dem kleinen Kugelfisch erreicht. Dies könnte zum einen auf die Rotation durch eine Kugel aber auch durch die unterschiedliche Translationstechnik die hier zum Einsatz kommt, zurück zuführen sein. Dies lässt sich jetzt nicht aus diesem Diagramm lesen und wird im Teil der Simultanitäten (Kapitel 6.2) genauer betrachtet.

Mit der Kugelmaus wurden die zweitbesten TCTs erzielt. Auch hier kommt eine Kugel zum Einsatz um die Rotation durchzuführen. Translation wird jedoch durch den SpaceMouse-Sockel erreicht (Kapitel 2). Die Tatsache, dass die Aufgabe mit kugelgesteuerter Rotationen schneller ausgeführt worden ist, als bei der SpaceMouse könnte man zumindest sagen, dass die Rotation mit der Kugel einen Gewinn an Performance bringt. Genaue Aussagen sind jedoch auch hier erst durch die Betrachtung der Simultanitäten möglich.

Auch die SA (Abb. 6.4) zeigt eine stetige Verbesserung der Performance. Die Personen scheinen mit zunehmendem Training, immer besser mit den Geräten zurechtzukommen. Die von den Personen erreichten TCTs werden konstanter, d.h. sie weichen immer weniger voneinander ab, was darauf zurückzuführen sein könnte, das die Versuchspersonen immer vertrauter im Umgang mit den Eingabegeräten werden.

Das die Kugelmaus in Session 2 eine geringere SA aufweist als der kleine Kugelfisch, ist eventuell darauf zurückzuführen, das die Versuchspersonen nach relativ kurzer Zeit schon recht gut mit der Kugelmaus umgehen können, weil sie von der Handhaltung her einer herkömmlichen Desktop Maus noch am nahesten kommt. Jedoch zeigt sich auch, dass in Session 3 beide SA's gleichauf und in Session 4 der kleine Kugelfisch wieder vorne liegt. Das bedeutet also, dass trotz der schnellen Eingewöhnungsphase bei der Kugelmaus, die konstanteren Zeiten bei dem kleinen Kugelfisch erreicht werden. Also scheint der kleine Kugelfisch weitere Vorteile als nur die Kugelrotation zu haben (vgl. Simultanitäten). Man kann jedoch auch erkennen das die Verbesserung der SA bei der Kugelmaus am Größten ausfällt (Reduzierung um 50%). Danach folgt der kleine Kugelfisch. Die Verbesserung der SA fällt bei der SpaceMouse am geringsten aus. Was evtl. darauf zurückzuführen ist, dass auf Grund der unnatürlichen Rotation mit dem Puck die Gewöhnung an das Gerät recht langsam vor sich geht.

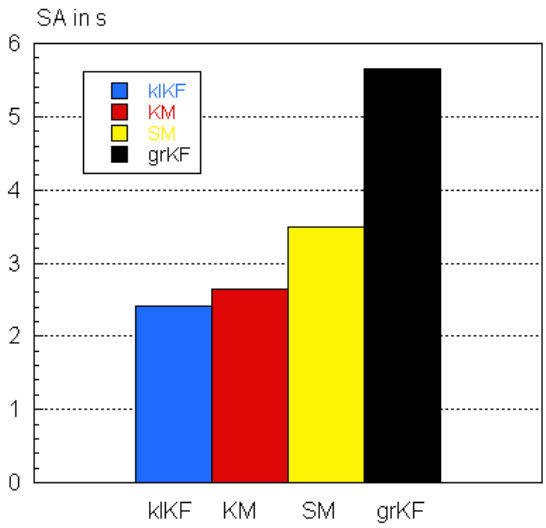

Abb. 6.5: Mittelwerte der SA über 4 Sessions gebildet.

In der Abbildung 6.5 sind die Mittelwerte der SA noch einmal einzeln dargestellt.

Anhand dieser Grafik wird noch erneut deutlich, dass die zwei Geräte mit Kugel eine geringere SA aufweisen als die SpaceMouse.

6.2. Simultanitäten

6.2.1 Problematik:

Wir haben die ausgelösten Rohwerte der Inputkanäle in den WriteOutFiles über alle Trialverläufe hinweg gespeichert (Kapitel 5.3.8). Nun versuchen wir daraus das Verhalten und die Intention der VP zu rekonstruieren und letztendlich gerätespezifische Vorgehensweisen bzw. Besonderheiten ableiten zu können.

Ein Ausschnitt aus dem WriteOutFile könnte z.B. so aussehen:

Frame	RX	RY	RZ
1	0,011	0	0,022
2	0,011	0	0,033
3	0,011	0,033	0,011
4	0,011	0	0
5	0,011	0	0,022

Der Kanal RX weist über mehrere Frames einen konstanten Input auf. Dies könnte bedeuten, dass die Versuchspersonen bewusst eine Rotation vorgenommen hat. Andererseits sind die Inputs hier sehr klein und könnten als unbewusster Nebeneffekt eines anderen Kanals auftreten.

RY zeigt lediglich einen Eintrag in diesen Frames. Zu vermuten ist, dass solche kurzen Inputs (1 bis 2 Frames) ungewollt vom Benutzer sind und durch motorische Unruhe oder andere Streueffekte entstehen. Mit steigendem Betrag solcher Ausschläge muss jedoch von bewussten Inputs ausgegangen werden.

Häufig sind Inputs in der Form von RZ anzufinden. Die Werte wechseln oft und können auch durch kurze Nullfolgen unterbrochen werden.

Auch bei bewusst gleichmäßig durchgeführten Aktionen, kann durch die hohe Abtastfrequenz der Sensorik eine unregelmäßige Wertefolge entstehen. Selbst

kleinste Abweichungen und Vibrationen der Hand können wahrgenommen werden und zu unsteten Wertefolgen führen.

Nun ist es sehr schwierig auf diese vielen Parameter und Eventualitäten einen einheitlichen Bewertungssatz aufzustellen.

Eine Möglichkeit Streueffekte zu eliminieren ist der Einsatz von zusätzlichen Filtern.

Wir versuchen möglichst schwache Filter zu verwenden, da bereits in der Testapplikation die stärksten Streuinputs durch Translations- und Rotationsschwellwerte gefiltert wurden. Außerdem haben die Versuchspersonen mit diesen Parametereinstellungen getestet und gelernt mit den gerätespezifischen Besonderheiten zu Recht zu kommen.

Im folgenden Absatz werden mehre Filtervarianten vorgestellt und diskutiert welcher davon für die Simultanitätsbetrachtung verwendet werden soll.

6.2.2 Datenbereinigung/-filterung

Für die Simultanitätsbetrachtungen müssen unsere Datensätze weiter bereinigt und gefiltert werden. Der Datensatz von Versuchsperson 1 wurde nicht in die Betrachtung mit einbezogen. Die Gründe hierfür sind dieselben wie im Kapitel 6.1 beschrieben.

Wir unterteilen die Aktionen der Versuchspersonen in vier verschiedene Simultanitätskategorien.

Dies sind die Kategorien „Nur Rot“, „Nur Trans“, wo jeweils entweder nur Rotation oder nur Translationen ausgeführt worden sind. Die anderen beiden Kategorien sind „Nichts“, wo überhaupt keine Aktionen seitens des User ausgeführt wurden und „Beides“, wo Rotation und Translation gleichzeitig also simultan benutzt wurden.

Im Kapitel 4.5 wurde beschrieben, dass am Ende jedes Trials, wenn die Dockingposition erreicht wurde, der Tetraeder für 0.8 Sekunden weiter im Zielbereich gehalten werden muss. Erst anschließend gilt der DockingTask als erfüllt. Dies stellt sicher, dass die Zielposition willentlich erreicht worden ist und nicht nur zufällig „durchfahren“ wurde. In diesen 0.8 Sekunden Latenzzeit wird die Zielposition lediglich gehalten. Es werden also keine bedeutenden Inputs mehr getätigt, im

größten Teil dieses Zeitabschnitts finden gar keine Interaktionen mit dem Gerät mehr statt. Der „Nichts“ -Anteil erhöht sich demzufolge und entspricht außerdem nicht mehr seiner ursprünglichen Bedeutung, nämlich der Strategieplanung und Planung der Interaktion mit dem Eingabegerät. Da im Test TCTs von weniger als 8 Sekunden erreicht werden, entsprechen die 0.8 Sekunden Latenzzeit immerhin einem Anteil von 10% und können die Simultanitätsbetrachtung bedeutend verfälschen.

Wir haben deshalb diese Latenzzeit vom Ende aller Trials abgezogen. Die Resultate sind jeweils in der linken oberen Grafik (Abb. 6.6, 6.10, 6.14) zu sehen.

Es ist zu sehen, dass die Simultanitätskategorie „Beides“ bei SpaceMouse und dem kleinen Kugelfisch einen hohen bzw. bei der SpaceMouse sogar den stärksten Anteil ausmacht. Der kleine Kugelfisch unterstützt zwar von seiner Konstruktion her den gleichzeitigen Einsatz von Rotation und Translation, aber Simultanitätsanteile von bis zu 38% decken sich nicht mit dem beobachteten Simultanitätsverhalten der Benutzer bei diesem Gerät. Auch bei der SpaceMouse ist ein Simultanitätsanteil von bis 55% in Session 4 fragwürdig. Allerdings kann dieser Anteil auch durch unbeabsichtigte Streuinputs entstehen.

Zwischen dem reinen Translationsfilter 0,8mm/sec und dem Filter Translation 0.8mm/sec & Rotation 5°/sec, erfährt der „Beides“–Anteil eine Reduzierung um 10 bis 15 Prozentpunkte, was für diese Kategorie umgerechnet eine Verringerung von 30% darstellt. Ein großer Anteil der „Beides“–Kategorie besteht also aus Rotationsinputs <5°/sec und könnte ungewollt zustande gekommen sein, z.B. indem beim Translieren unabsichtlich eine kleine Rotation mit ausgelöst wurde.

Die hohen Simultanitätsanteile könnten also auch auf unzureichende Trennungsmechanismen zwischen Translation und Rotation zurückzuführen sein.

Die Schwellwerte von Translation und Rotation könnten zu gering angesetzt worden sein. Wir können jetzt für alle Datensätze zunächst den Translationsthreshold etwas erhöhen.

Dabei muss jedoch sehr vorsichtig vorgegangen werden, um keine gerätetypischen Simultanitätseffekte wegzufiltern. Zunächst haben wir einen Translationsfilter von 0.8mm/sec ausprobiert, d.h. alle Bewegungen <0,8mm/sec werden ignoriert also als

„Nichts“-Anteil interpretiert. Die Filterergebnisse sind jeweils in der oberen rechten Grafik zu sehen (Abb.: 6.7, 6.11, 6.14).

Der „Beides“-Anteil bei der SpaceMouse und dem kleinen Kugelfisch erfährt so eine Verringerung um ca. 5 Prozentpunkte. Diese 5 Prozentpunkte schlagen sich Anteilweise auf die „Nichts“–Kategorie nieder, die sich demzufolge leicht erhöht.

Im nächsten Schritt setzen wir neben dem Translationsfilter weiterhin noch einen Rotationsfilter ein. Die Wahl des Rotationsthresholds gestaltet sich jedoch als schwierig. Die kleinste Auflösungsstufe der Kugelsensorik (Kuglemaus und kleiner Kugelfisch) beträgt nur ca. 5°/sec, bei der SpaceMouse ist die Auflösung jedoch wesentlich genauer, umgerechnet <1°/sec. Dies erscheint beim Betrachten des sehr guten Abschneidens der Kugelgeräte (Kapitel 6.1) als sehr verwunderlich. Die Erklärung hierfür liegt in der Positionssteuerung der Rotation bei den Kugelgeräten. Es werden pro Sekunde 42mal die Bewegungen der Kugel abgetastet. Jeder Messzyklus erreicht daher eine Genauigkeit von 0,12°/sec (5/42). Durch die Positionssteuerung der Kugel können die benötigten hohen Genauigkeiten durch Inputs von wenigen Messzyklen (Sekundenbruchteilen) erreicht werden.

Wir möchten keine unterschiedlichen Rotationsthresholds für die drei Geräte einsetzten, also probieren wir zunächst das kleinste gemeinsame Vielfache der drei Geräte, nämlich 5°/sec. Jeweils die linke untere Grafik (Abb.: 6.8, 6.12, 6.15) der folgenden Seiten zeigt diese Filtervariante.

Es zeigt sich, dass der „Beides“–Anteil bei der SpaceMouse und des kleinen Kugelfisches sehr stark reduziert wurde, um bis zu 20 Prozentpunkte. Beim kleinen Kugelfisch wurde sogar die Simultanitätslernkurve (Verringerung von „NurRot“ bei gleichzeitiger Erhöhung von „Beides“) nahezu komplett weggefiltert, ein deutliches Anzeichen für einen zu starken Rotationsfilter. Außerdem steigt der „Nichts“ -Anteil deutlich an und liegt im Mittel über 25%.

Es erscheint ziemlich unrealistisch, dass mehr als ¼ der Zeit mit Strategie- und Geräteplanung verbracht wird. Dies deckt sich nicht mit unseren Beobachtungen der Probanden. Einen Planungsanteil von ca. 20% halten wir für realistisch und versuchen dies bei der Wahl der Filter zu beachten.

Der Rotationsfilter von 5°/sec ist also zu hoch angesetzt. Die Feineinstellungen zum erreichen der Dockingtoleranz erfordern besonders die kleineren Inputs der Geräte. Gerade diese hohen Genauigkeiten werden bei einer Schwelle von 5°/sec zum großen Teil weggefiltert.

Wie weiter oben geschildert, sind diese 5°/sec die kleinste Auflösungsstufe der Kugelsensorik aber dennoch zu stark für unsere Anforderungen.

Mit unserem Auswertungstool SPSS können wird jedoch eine noch kleinere Auflösungsstufe der Kugel simulieren. Der kleinste Input der Kugelsensorik beträgt 0,011 und entspricht wie erwähnt einer Genauigkeit von 0,12°. Bei aufeinander folgenden Wertereihen von 0,011 wird jeder zweite Wert ignoriert.

Frame	RX	Grad/Frame	RX (gefiltert)	Grad/Frame
1	0,011	0,12°	0,011	0,12°
2	0,011	0,12°	0	0
3	0,011	0,12°	0,011	0,12°
4	0,011	0,12°	0	0
5	0,011	0,12°	0,011	0,12°
Summe		0.6°		0,36°

Durch Filterung jedes zweiten Werts, beträgt die Genauigkeit hochgerechnet ca. 2,5°/sec. Sicherlich ist diese Vorgehensweise nicht optimal, aber der reguläre Rotationsfilter von 5°/sec entfernt einen Großteil der Simultanitätscharakteristiken sowie das Feindocking. Der reine Translationsfilter ist dagegen zu schwach den Datensatz zu „säubern“. Vibrationen und motorische Unruhe, bei der Steuerung der Kugel, verfälschen die Daten.

Deshalb erscheint ein schwacher kombinierter Filter mit Translationsschwelle <0,8mm/sec und Rotationsschwelle <2,5°/sec uns als geeignet und soll letztendlich auch für die weiter Simultanitätsbetrachtung verwendet werden. Dieser Filter ist jeweils in der rechten unteren Grafik zu sehen (Abb.: 6.9, 6.13, 6.17).

Ein alternativer Ansatz zur Bewertung der Effizienz der Inputs basiert auf der MMetric von Masliah. Dabei wird der aktuell gegangene Weg mit dem optimalen Weg in Relation gesetzt und daraus ein Effizienzmaß berechnet.

Bei uns würden dann nur solche Frames betrachtet, welche die Diskrepanz zum Ziel verringern, der Rest könnte ignoriert werden.

Für die Translation kann die euklidische Distanz von Target und Cursor in jedem Frame bewertet werden. Bei kleiner werdender Distanz werden die Translationsinputs gewertet, steigt sie dagegen werden sie verworfen.

Schwieriger gestaltet sich die Bewertung der Orientierungsdiskrepanz, da ein Wegdrehen ab einem gewissen Drehwinkel wieder als Annäherung interpretiert werden kann. Es ist schwierig alleine aus den Daten zielgerichtete von fehlgerichteten Rotationen zu unterscheiden. Außerdem kann auf einem Rotationsanteil RX, RY, RZ eine Annäherung stattfinden, die anderen Anteile sich dagegen entfernen. Ein von Rotations- und Achsenanteilen unabhängiges Modell ist daher von Nöten, z.B. das Quaternionen-Modell.

Dieser Ansatz kann mit unserem WriteOut–Ansatz nicht ohne erheblichen Aufwand umgesetzt werden, da wir untereinander abhängige Eulerwinkel herausgeschrieben haben. Er ist aber als Ansatzpunkt für weitere Auswertungen in Betracht zu ziehen.

Auf den nächsten Seiten folgen die genannten Filtergrafiken und eine Beschreibung gerätespezifische Beschreibung der auftretenden Effekte. Jeweils 4 Grafiken pro Gerät/Seite.

Filtereffekte Kugelmaus

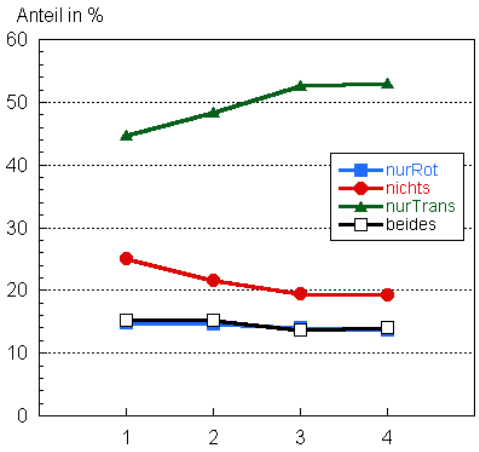

Abb 6.6.: KM1- ohne 0.8sec Latenzzeit am Ende des DockingTask

Der „Beides“–Anteil sollte theoretisch nicht vorhanden sein, da zwischen Rotation und Translation umgegriffen werden muss. Der „Nichts“ -Anteil ist relativ hoch, da hier die Umgreifzeit zwischen Kugel und Sockel enthalten ist.

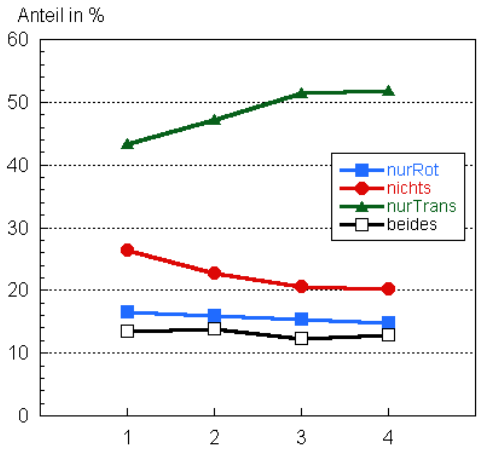

Abb 6.7: KM2- Translationsfilter 0.8mm/sec.

„NurTrans“ und „Beides“ geringfügig kleiner, dafür „Nichts“ und „NurRot“ minimal höher. Insgesamt hat dieser Filter nur einen sehr geringen Effekt auf die KM.

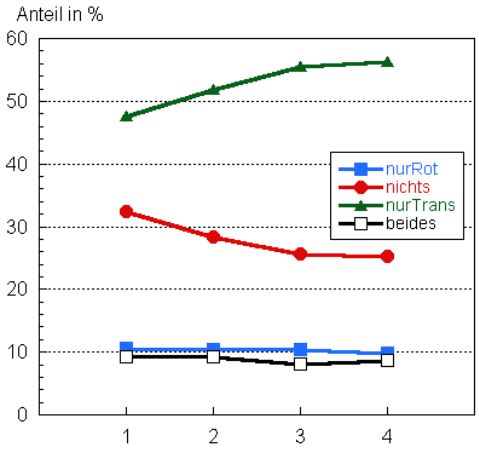

Abb. 6.8: KM3- Translationsfilter 0.8mm/sec und Rotationsfilter 5°/sec.

„Nichts“–Anteil um 6 Prozentpunkte angestiegen; trotz enthaltener Umgreifzeit jetzt relativ hoch. „NurRot“ um ca. 6 Prozentpunkte gesunken. Rotationsfeindocking stark weggefiltert. „Beides“ wie gewünscht sehr niedrig, <10%.

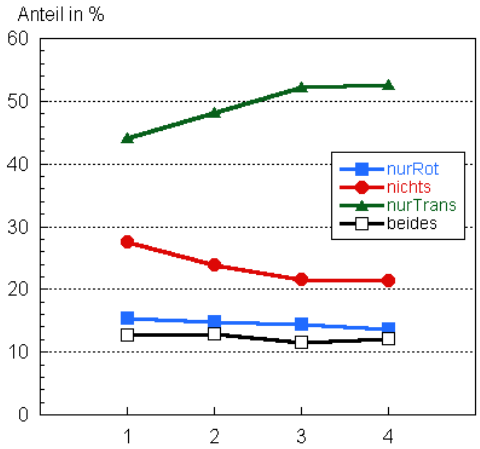

Abb. 6.9: KM4- Translationsfilter 0.8mm/sec und Rotationsfilter 2,5°/sec.

Rotationsfeindocking noch erhalten, da „NurRot“ hier wieder deutlich höher als bei Rotationsfilter 5°/sec.

Filtereffekte kleiner Kugelfisch

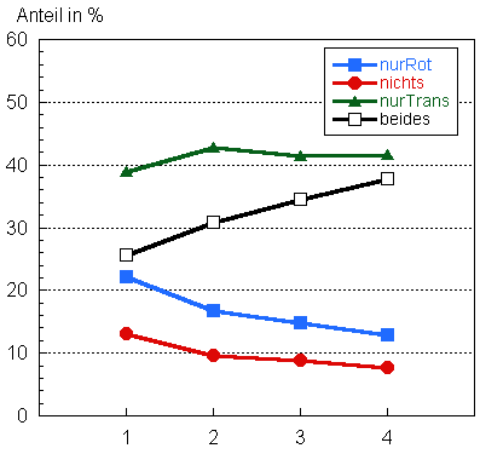

Abb. 6.10: KlKF1– ohne 0.8sec Latenzzeit am Ende des Docking Task.

Die ca. 10% Verringerung über 4 Blöcke in „NurRot“ werden fast vollständig auf „Beides“ übertragen. Simultanes Arbeiten wird also erlernt. Dieser Lernerfolg sollte auch nach Filterung noch vorhanden sein! „Beides“ erscheint mit >30% hier aber unrealistisch hoch.

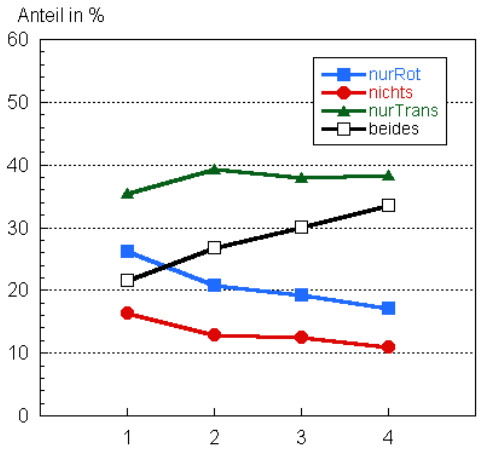

Abb. 6.11: KlKF2- Translationsfilter 0.8mm/sec.

„Beides“ um ca. 5Prozentpunkte reduziert. „Nur Trans“ minimal verringert. Gewonnene Prozentpunkte gleichmäßig auf „Nur Rot“ und „Nichts“ verteilt; jeweils plus 2-3Prozent. Lernerfolg bezüglich Simultanität noch erkennbar

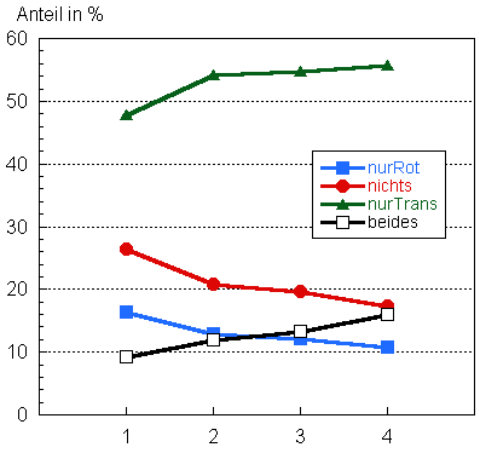

Abb. 6.12: KlKF3- Translationsfilter 0.8mm/sec & Rotationsfilter 5°/sec

„Beides“–Anteil stark reduziert. Lernerfolge von „NurRot“ und „Beides“ betragsmäßig stark reduziert. Feindocking nahezu komplett wegfiltert. „Nichts“ mit ca. 20% sehr hoch.Filter erscheint sehr stark.

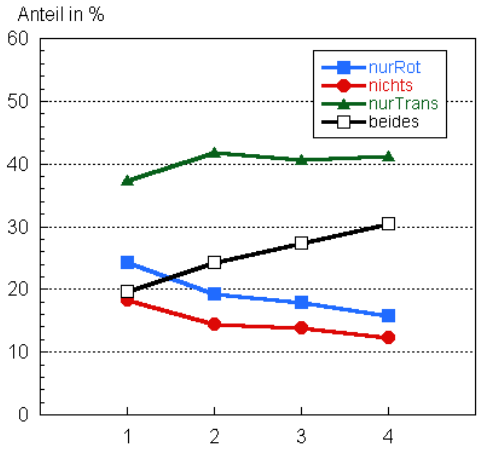

Abb. 6.13: KlKF4- Translationsfilter 0.8mm/sec & Rotationsfilter 2,5°/sec

Gegenläufigkeit von „NurRot“ und „Beides“ wieder deutlich erkennbar. „Nichts“ wieder in realistischere Bereiche reduziert. Feindocking mit diesem Filter noch enthalten.

Filtereffekte SpaceMouse

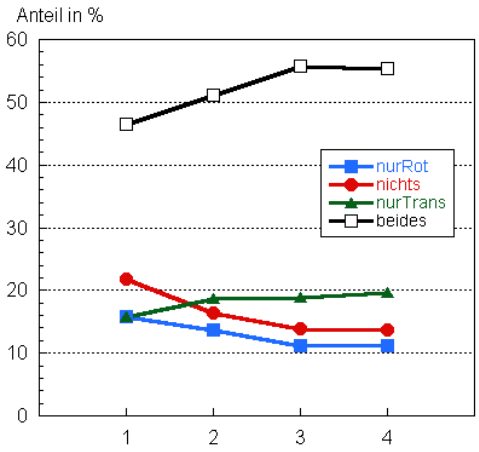

Abb. 6.14: SM1- ohne 0.8sec Latenzzeit am Ende des Docking Task.

„Beides“ zeigt starke Verbesserung in den ersten 3 Blöcken, ist aber insgesamt unrealistisch hoch. „Nichts“ Reduzierung um ca. 10% über 4Sessions. Die freiwerdenden Kapazitäten tragen mit zum Anstieg der „Beides“ Kategorie bei.

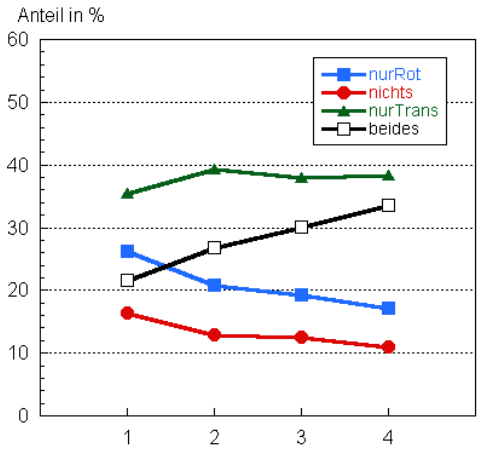

Abb. 6.15: SM2- Translationsfilter 0.8mm/sec.

Die Form der Kurven bleibt erhalten. Es verändern sich lediglich die Niveaus. Der „Beides“ –Anteil erfährt eine Reduzierung um ca. 6Prozentpunkte. Im Gegenzug steigen „Nichts“ und „NurRot“ –Anteile.

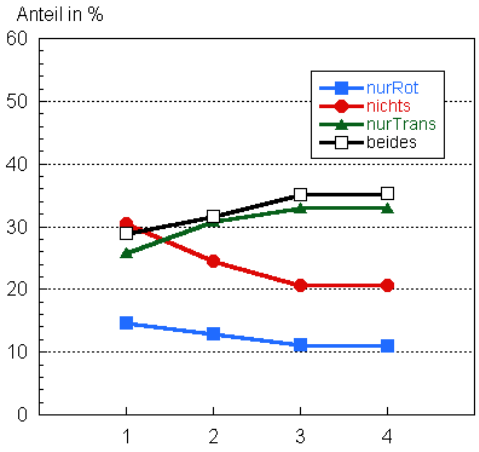

Abb. 6.16: SM3 - Translationsfilter 0.8mm/sec & Rotationsfilter 5°/sec

„Beides“ stark reduziert und Anstieg eingeebnet; Block 1&2 minus 6-7%, Block 3&4 minus 15%. „Beides“ beruht offensichtlich stark auf kleinen (<5°/sec) Rotationsanteilen. Diese kleinen Anteile könnten unabsichtlich bei Translationsvorgängen mit ausgelöst worden sein.

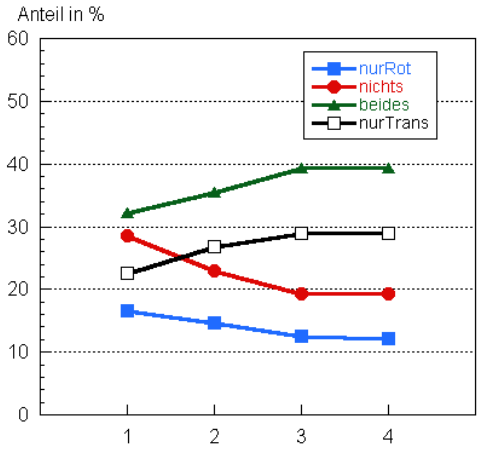

Abb. 6.17: SM4 - Translationsfilter 0.8mm/sec & Rotationsfilter 2,5°/sec

„Beides“ wieder höher. Daher müssen viele kleine Rotationsanteile in „Beides“ enthalten sein. „Nichts“ -Anteil wieder etwas kleiner erscheint aber immer noch sehr hoch.

Wir wollen unsere Haupthypothese „*Rotationen mittels Positionssteuerung einer Kugel sind intuitiver/einfacher zu steuern, als geschwindigkeitsgesteuerte Rotationen der SpaceMouse. Das Arbeiten gestaltet sich so bedeutend effektiver!*“ anhand dieser Daten und unter dem Aspekt der Simultanität überprüfen.

Des weiteren können simultanitätsspezifische Thesen aufgestellt werden:

6.2.3 Simultanitätshypothesen

1. *Beim kleinen Kugelfisch wird Simultanität gelernt.*
2. *Bei der SpaceMouse wird Separierbarkeit gelernt.*
3. *Umgreifen führt bei der Kugelmaus im Vergleich zum kleinen Kugelfisch zu schlechterer Performance.*

Zu Simultanitätshypothese 1:

Der kleine Kugelfisch unterstützt von seiner Konstruktion her den gleichzeitigen Einsatz von Rotation und Translation. Sowohl Translation als auch Rotation werden an der Kugel ausgeführt. Es ist kein Umgreifen notwendig.

Zu Simultanitätshypothese 2:

Bei der SpaceMouse ist es schwierig die geplanten Aktionen auch so auf das Gerät zu übertragen. Wenn man z.B. nicht genau entlang einer Achse verschiebt, können unbeabsichtigte Eingaben von anderen Inputkanälen z.B. Rotationen auftreten und letztendlich die Abweichung zum Zieltetraeder sogar noch vergrößern. Diese Streuinputs müssen im weiteren Verlauf des Trials ausgeglichen werden, was zusätzlich Zeit kostet.

Die Versuchspersonen werden sicherlich lernen diese Streuinputs zu minimieren, indem sie z.B. ihre Griffposition am Sockel der SpaceMouse optimieren und sich häufiger entlang der Geräteachsen bewegen. Wir bezeichnen das Herauslösen bestimmter DoFs und DoF-Kombinationen ohne Streuinputs auf andren (benachbarten) Inputkanälen, als Separation von Freiheitsgraden.

Zu Simultanitätshypothese 3:

Umgreifen stellt einen zeitlichen Mehraufwand dar.

6.2.3.1 Hypothesenprüfung anhand der Anteile der einzelnen Simultanitätstypen

Die Darstellungen der folgenden Seiten zeigen in der oberen Grafik jeweils den Prozentanteil unserer vier Simultanitätskategorien (nur Translation, nur Rotation, Translation & Rotation, keine Aktivität) an der mittleren TCT.

Die mittlere Grafik zeigt den absoluten Zeitbetrag der vier Simultanitätskategorien. Diese Teilzeiten summieren sich letztendlich zur mittleren TCT auf.

In der untersten Grafik wurden nur die bewussten Aktionen miteinbezogen, d.h. der „Nichts“-Anteil wurde ignoriert. Demzufolge ergeben sich neue Prozentanteile für die Kategorien „NurRot“, „NurTrans“ und „Beides“. Diese Anteile summieren sich insgesamt zu 100% auf.

In jedem Session stehen drei Balken nebeneinander. Der mittlere Balken enthält die eben beschriebenen Prozentanteile. Links und rechts davon stehen die Balken GesTrans und GesRot. Diese Balken entsprechen dem gesamten Rotations- bzw. Translationsanteil innerhalb der Session, d. h. für Rotation ist sowohl der „NurRot“–Anteil als auch der „Beides“-Anteil einbezogen, da in „Beides“ ja auch Rotationen enthalten sind.

Für alle Abbildungen wurde der Filter: Translation 0,8mm/sec & Rotation 2,5°/sec eingesetzt.

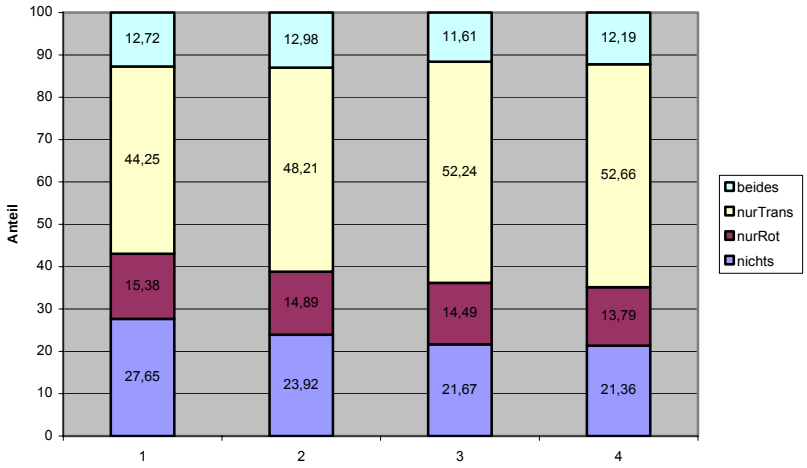

Abb.6.18: Kugelmaus Prozentanteile

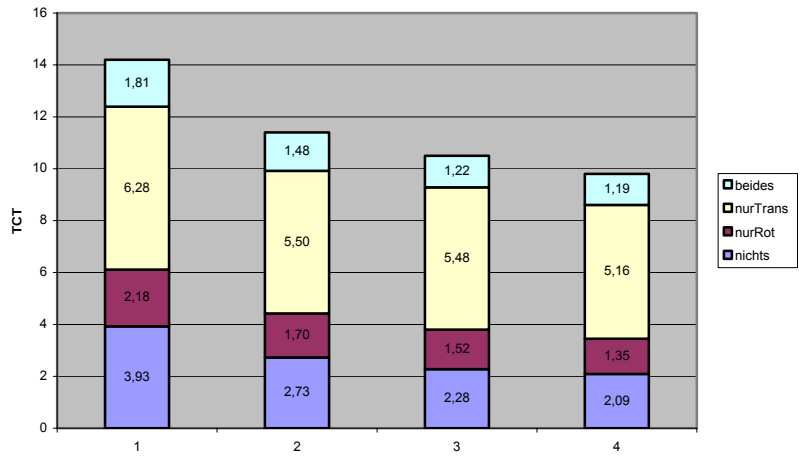

Abb 6.19: Kugelmaus Zeitanteile

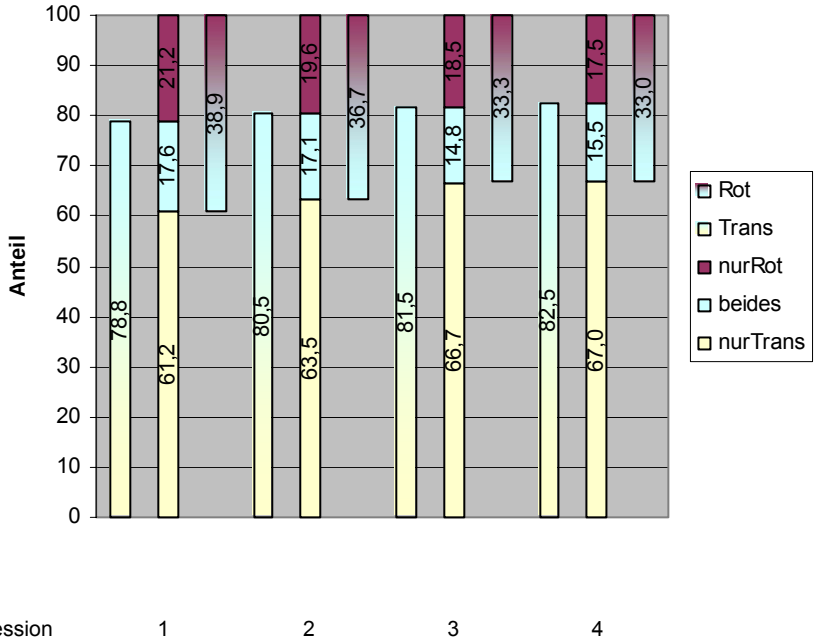

Abb. 6.20: Prozentanteile ohne „Nichts“-Kategorie

6.2.3.1.1 Kugelmaus

Auffällig ist im Vergleich mit den anderen Geräten der erhöhte „Nichts“-Anteil, was aber durch den zusätzlichen zeitlichen Aufwand beim Umgreifen zwischen Sockel und Kugel erklärt werden kann. Dieser Anteil erfährt von Session 1 (27,6%) zu Session 4 (21,4%) eine Reduzierung um ca. 6 Prozentpunkte, was absolut einer Verringerung um 2 Sekunden entspricht (von 4 sec in Session 1 zu 2 sec in Session 4). Dies stellt eine Verringerung um 50% dar.

Dies bedeutet, dass die Geschwindigkeit des Umgreifprozesses verbessert und/oder die Häufigkeit des Umgreifens verringert werden konnte.

Wie aus dem TCT Kapitel 6.1 bekannt, schneidet der direkte Konkurrent kleiner Kugelfisch 1 bis 1,5 Sekunden besser ab.

TCT	1	2	3	4
KIKF	13,4	10,5	8,9	8,3
KM	14,2	11,4	10,5	9,8

Die „Nichts“–Zeiten beim kleinen Kuglefisch sind genau um diese 1 bis 1,5 Sekunden geringer.

Das geringfügig schlechtere Abschneiden der Kugelmaus gegenüber dem kleinen Kugelfisch kann also auf den zusätzlichen zeitlichen Aufwand beim Umgreifen zurückgeführt werden. Die Simultanitätshypothese 3 kann also als gültig angesehen werden.

Der „Beides“–Anteil sollte, durch das Umgreifprinzip bei der Kugelmaus, eigentlich nicht vorhanden sein. Er ist aber konstruktionstechnisch bedingt nicht komplett vermeidbar. Die oberste Grafik zeigt, dass der „Beides“–Prozentanteil über alle vier Sessions nahezu konstant bleibt.

Des weiteren wissen wir, dass „Beides“ nur bei Manipulation der Kugel auftritt, wo durch Vibrationen und das Eigengewicht der Hand, ungewollte Inputs auf die

Translationssensoren ausgeübt werden. Der „Beides“-Anteil könnte demzufolge komplett als Rotation interpretiert werden.

In der untersten Grafik ist zu erkennen, dass sich der GesamtRotations-Anteil über die 4 Sessions von 39% auf 33% verringert. Absolut entspricht dies einem Zeitanteil von ca. 4 sec im Session 1 und 2,5 sec in Session 4, was einer Verringerung um 40% darstellt (siehe mittlere Grafik).

Da sich die Startorientierungen der Tetraederpositionen ständig wiederholen erkennt man, dass die nötigen Rotationsvorgänge zum Docken über die Sessions hinweg effizienter ausgeführt werden.

Rotationssteuerung über Positionskontrolle der Kugel ist also schnell und verbesserbar.

Im Gegensatz dazu bietet die Translation nur ein geringes Verbesserungspotential. Zwar verringert sich die absolute Translationszeit um 1,1 sec von Session 1 zu 4, aber bei insgesamt geringeren TCTs steigt der Translationsanteil von 44% in Session 1 um 9 Prozentpunkte auf 53% in Session 4. In den Sessions 3 und 4 nimmt die Translation sogar die Hälfte der Trialzeit in Anspruch.

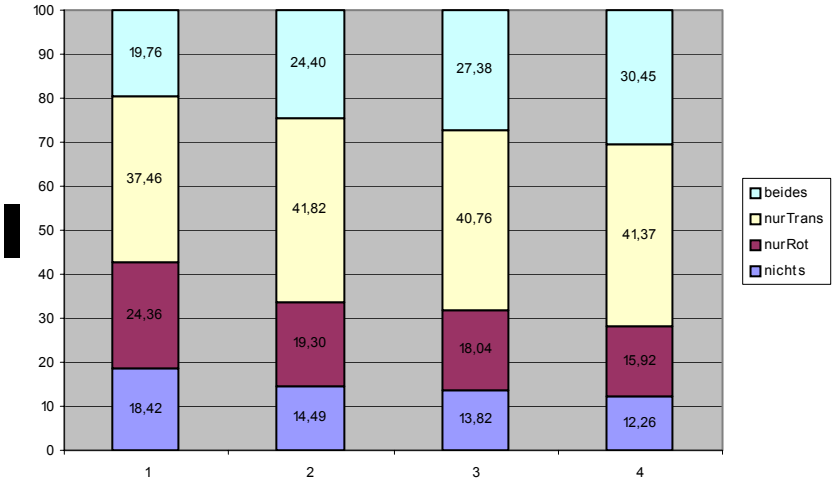

Abb.6.21: Kleiner Kugelfisch Prozentanteile.

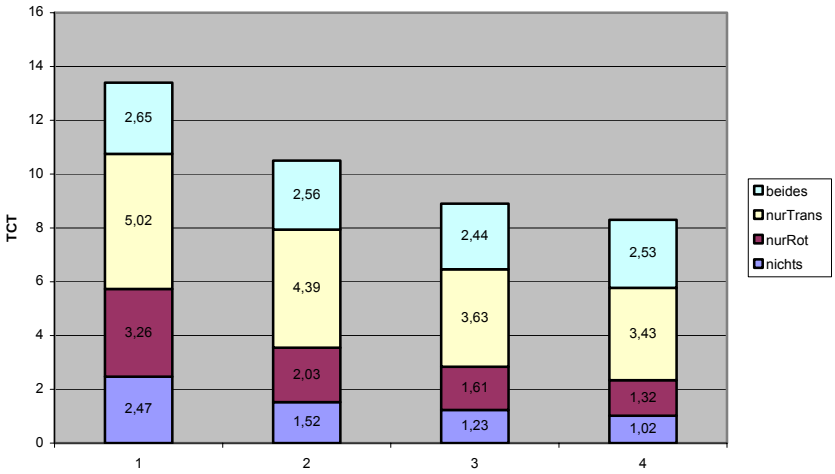

Abb.6.22: Kleiner Kugelfisch Zeitanteile.

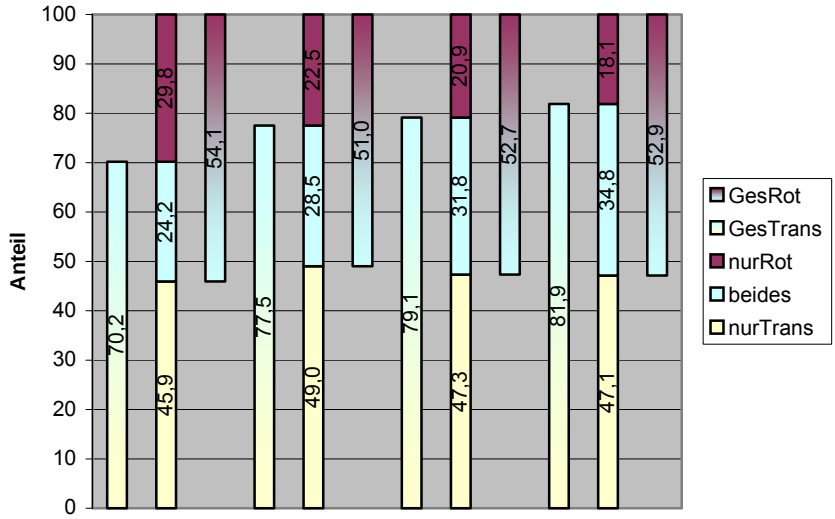

Abb. 6.23: Prozentanteile ohne „Nichts“-Kategorie

6.2.3.1.2 Kleiner Kugelfisch

Zunächst wird die unterste Grafik betrachtet. Der „Beides"–Anteil erhöht sich hier um ca. 11 Prozentpunkte von Session 1 zu Session 4. Er startet bei 24% und liegt am Ende bei 35%. Der „NurRot"–Anteil verringert sich dagegen um ca. 12% über die 4 Sessions. In Session 1 liegt er bei 30% und in Session 4 bei 18%. In der obersten Grafik zeigen sich diese Effekte ebenfalls sehr deutlich. Der GesamtTranslations – Anteil (siehe unterste Grafik) steigt über 4 Sessions von 70% auf 82%. Der „NurTrans"-Anteil bleibt dagegen konstant.

„NurRot"↓	„GesRot" ↔
„Beides" ↑	
„NurTrans" ↔	„GesTrans" ↑

Hier findet ein Transfer von Anteilen statt. Translation werden vermehrt mit Rotationen verbunden. D.h. Simultanität zwischen Translation und Rotation wird bei diesem Gerät gelernt. Die 1. Simultanitätshypothese kann für den kleinen Kugelfisch bestätigt werden.

Die Effizienz dieser Simultanitäten muss jedoch überprüft werden. Dazu werden erneut die TCTs der Geräte benötigt.

TCT	1	2	3	4
KIKF	13,4	10,5	8,9	8,3
KM	14,2	11,4	10,5	9,8

Der kleine Kugelfisch schneidet 1 bis 1,5 sec besser als die Kugelmaus ab. Wie bei den Ergebnissen der Kugelmaus zu sehen, kann dieser Unterschied genau durch die Umgreifzeit der Kugelmaus erklärt werden. Lässt man diesen Nachteil außer Acht, bedeutet dies, dass beide Geräte gleich effizient sind. Und das bei unterschiedlichen Simultanitätsstrategien. Die Kugelmaus erlaubt keine Simultanität von Translation und Rotation und ist dennoch genauso effizient wie der kleine Kugelfisch. D.h. die

Trennung von Translation und Rotation führt dazu, dass beide Bereiche für sich alleine viel genauer kontrolliert und präziser eingesetzt werden können.

Der kleine Kugelfisch dagegen unterstützt vom Gerätedesign Simultanität von Translation und Rotation. Dieser Vorteil wird von den Versuchspersonen auch angenommen und gelernt, allerdings auf Kosten der Genauigkeit dieser Verbindung. Letztendlich hebt sich Simultanität mit der Genauigkeit (der Simultanität) auf und kann sich nicht als Vorteil ausspielen.

Die konstant hohen „GesRot"–Anteile beim kleinen Kugelfisch stützen diese Aussage. Sie liegen bei 51 bis 54%. Bei der Kugelmaus ist dieser Anteil nur 33 bis 39% groß. In Absoluter Zeit sieht es ähnlich aus.

„GesRot" beim kleinen Kugelfisch benötigt 4 bis 6 sec, bei der Kugelmaus dagegen nur 1,5 bis 4 sec.

Anmerkung:

Die GesamtRotations–Zeiten beim kleinen Kugelfisch sind immer noch besser als die entsprechenden Zeiten der SpaceMouse. Der Vorteil des kleinen Kugelfischs gegenüber der SpaceMouse, liegt aber vermutlich nicht in der bewussten Simultanität beim kleinen Kugelfisch, sondern in der Positionssteuerung der Rotation (siehe auch Ergebnisse Kugelmaus und SpaceMouse).

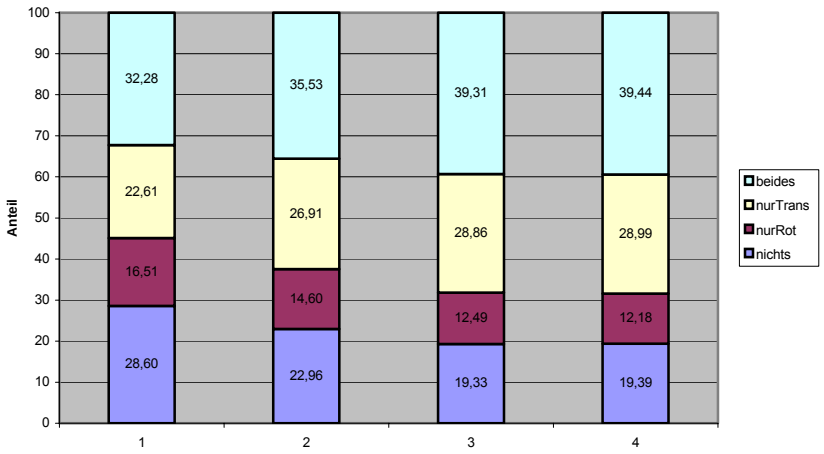

Abb. 6.24: SM Prozentanteil.

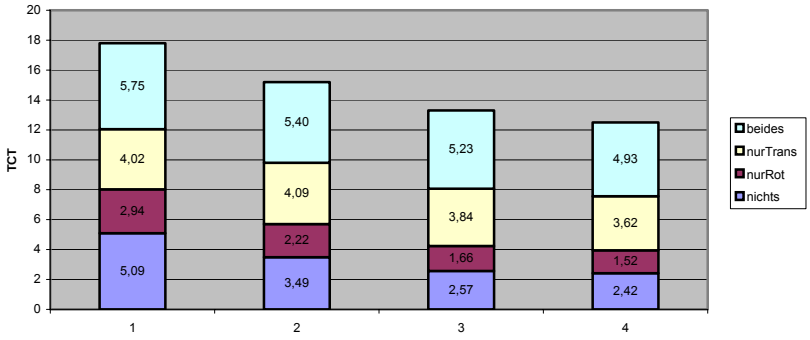

Abb. 6.25: SM Zeitanteil.

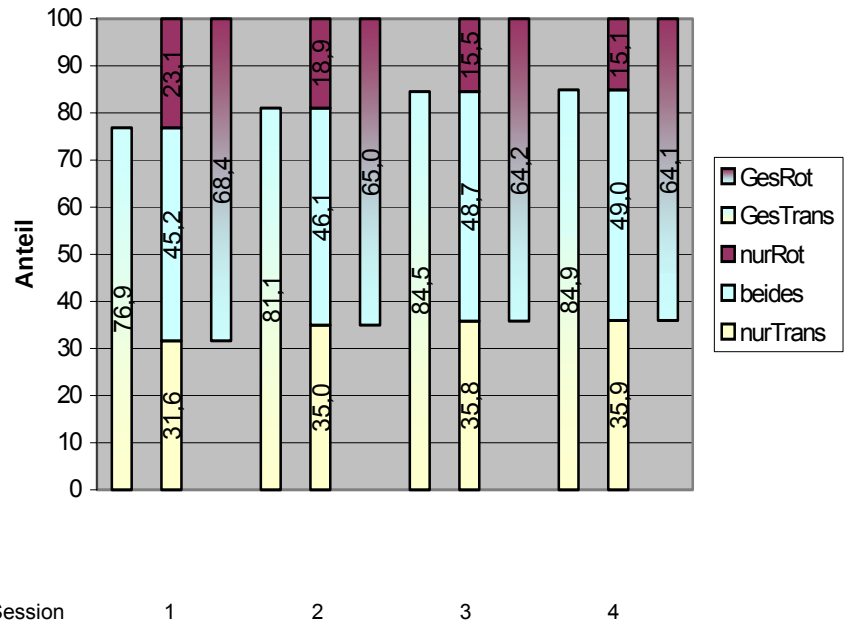

Abb. 6.26: Prozentanteile ohne „Nichts“-Kategorie

6.2.3.1.3 SpaceMouse

Der „GesRot“–Anteil in der unteren Grafik kann über die 4 Sessions hinweg leicht reduziert werden. Dennoch liegt dieser Anteil mit 64% in Session 4 rund doppelt so hoch wie der „GesRot“–Anteil in Session 4 der Kugelmaus (33%).

Der „GesRot“–Anteil in absoluter Zeit ist in Session 1 7 Sekunden, in Session 4 beträgt er 5 Sekunden. Zum Vergleich hier noch einmal die Absolutzeiten, die die Kugelmaus bzw. der kleine Kugelfisch für dieselben Rotationsvorgänge benötigen:

- Kugelmaus: Session 1: 4 sec, Session 4: 2,5 sec
- kleiner Kugelfisch: Session 1: 6 sec, Session 4: 4 sec

Dies ist ca. 60% schlechter als bei der Kugelmaus und ca. 30% schlechter als beim kleinen Kugelfisch. Dieselben Rotationsvorgänge können also weniger effizient als mit den Kugelgeräten vorgenommen werden.

Die wichtigste These unserer Studie, dass Positionsgesteuerte Rotation durch Drehen einer Kugel gegenüber Geschwindigkeitssteuerung der Rotation überliegt, bewahrheitet sich hiermit.

Von allen drei Geräten zeigt die SpaceMouse sowohl im Prozentanteil als auch in absoluter Zeit den größten „Beides“–Anteil. Man würde zunächst vermuten, dass dies für die SpaceMouse spricht. Wir haben beim kleinen Kugelfisch aber bereits gesehen, dass Simultanität nicht notwendigerweise auch schnelleres Arbeiten bedeutet, da die Genauigkeit von Simultanitäten nicht an die von Einzelaktionen herankommt.

Nun verbessern sich natürlich auch bei der SpaceMouse die TCTs über die Sessions hinweg.

TCT	**1**	**2**	**3**	**4**
SM	17,8	15,2	13,3	12,5

Wir möchten aber aus den Erfahrungen mit Genauigkeiten von Simultanitäten (siehe kleiner Kugelfisch) diese Verbesserungen nicht dem verstärkten Einsatz von Simultanitäten zuschreiben. Wir argumentieren lieber in der Art, dass die Versuchspersonen lernen die gewollten Aktionen auch tatsächlich mit dem Gerät umsetzen zu können, also die benötigten DoFs auch aus Sockelbewegungen herauszulösen. Separierbarkeit wird gelernt.

Diese Annahme lässt sich jedoch nicht allein auf Basis der bisher diskutierten Auswertungen bewerten. Es müssen noch weitere Untersuchungen vorgenommen werden.

Im Abschnitt „Simultanitäten der SpaceMouse“ wird noch einmal detaillierter auf dieses Thema eingegangen. Die 3. Simultanitätshypothese soll dort auch letztendlich beantwortet werden.

Sehr hoch sind bei der SpaceMouse außerdem die „Nichts“–Anteile, mit bis zu 5 sec, zu sehen in Session 1. Dieser Wert kann zwar über alle Sessions um 50% auf 2,5 sec reduziert werden, liegt aber dennoch höher als Session 4 der Kugelmaus (2 sec), wo immerhin auch noch Umgreifzeiten enthalten sind.

Dieser hoher Planungsanteil/Überlegungsanteil zur Gerätekontrolle, scheint notwendig zu sein, um gezielt Freiheitsgrade bzw. Freiheitsgradkombinationen aus den Sockelbewegungen separieren zu können.

6.2.4 Simultanitäten innerhalb 2DoF

Die folgende Abbildung zeigt die Häufigkeit aller möglichen Simultanitätskombinationen aus 2 DoF, im Vergleich (Abb. 6.27).

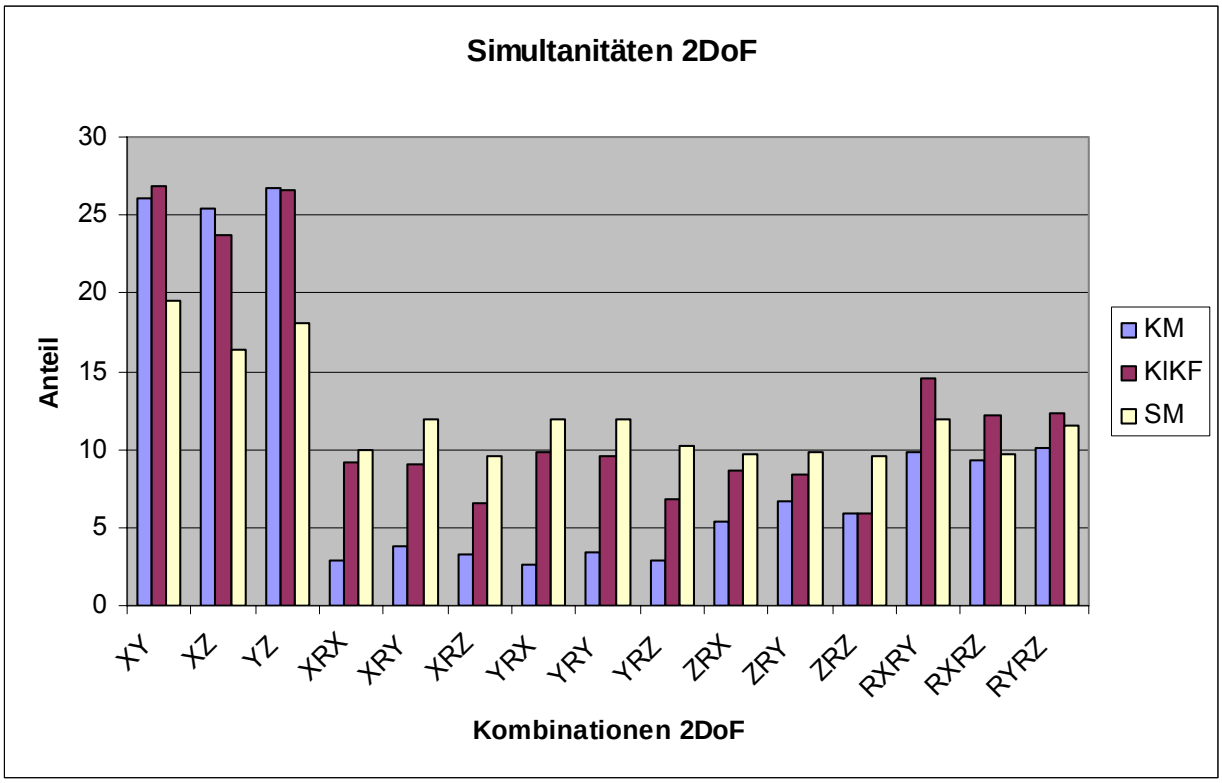

Abb. 6.27:

Vereinfachend kann man beim kleinen Kugelfisch von drei Gruppen sprechen, die auf unterschiedlichen Niveaus liegen. Die erste Gruppe stellen die reinen Translationssimultanitäten (linke 3 Balken) dar. Sie weisen die höchsten Anteile auf und liegen nahezu konstant bei 25%. Darauf folgt die Gruppe der reinen Rotationssimultanitäten (rechte 3 Balken) mit Anteilen von 13 bis 15%. Die letzte Gruppe sind die Simultanitäten aus einer Translation und einer Rotation (die mittleren Balken). Diese Gruppe erreicht insgesamt nur das niedrigste Anteilniveau.

Bei der Kugelmaus kann man eine ähnliche Gruppierung vornehmen; mit größtem Anteil die reinen Translationssimultanitäten mit ca. 25%. Es folgen die reinen Rotationssimultanitäten mit konstant 10% Anteil. Die Rotation-TranslationSimultanitäten sind bei der Kugelmaus konstruktionstechnisch nicht

vorgesehen aber bei unserem Kugelmaus–Prototyp nicht komplett vermeidbar gewesen. Sie werden hier nicht weiter betrachtet.

Die SpaceMouse ermöglicht alle diese drei Simultanitätsgruppen, zeigt aber kein deutliches gruppenspezifisches Anteilniveau. Die Verteilung ist im Vergleich zu den Kugelgeräten deutlich eingeebneter.

Besonders bei der Betrachtung der reinen Translationssimultanitäten (linke 3 Balken) fällt dies auf. Während die Kugelmaus und der kleine Kugelfisch über die ersten drei Balken nahezu identische Anteile aufweisen, fällt die SpaceMouse gegenüber diesen beiden Geräten deutlich ab.

Wir wissen, dass in allen drei Geräten die Translationen über eine SpaceMouse–Sensorik gemessen werden. Wie kommt es daher zu diesen geringeren Anteilen bei der SpaceMouse?

Vermutete Antwort:

Rotation und Translation der Kugelgeräte unterscheiden sich deutlich in ihrer Effizienz voneinander. Bei der SpaceMouse dagegen kann man von ähnlicher Effizienz von Translation und Rotation sprechen. In Anbetracht der Konstruktionsgrundlagen der SpaceMouse ist dies nicht verwunderlich, da Rotation und Translation mit derselben Messtechnik ermittelt werden. Sie werden also beide mit derselben „niedrigeren“ Effizienz ausgeführt. Mit den Kugelgeräten können die Orientierungsdiskrepanzen der beiden Tetraeder bedeutend schneller überwunden werden. Die Rotationsanteile an der Gesamtzeit sind demzufolge geringer als die Translationsanteile.

Rotation mittels Positionssteuerung einer Kugel ist deutlich zeiteffizienter als Rotationskontrolle durch Geschwindigkeitssteuerung, wie sie bei der SpaceMouse implementiert ist.

6.2.5 Simultanitäten innerhalb 3 DoF

Die folgende Abbildung zeigt die Häufigkeit aller möglichen Simultanitätskombinationen aus 3 DoF, im Vergleich (Abb. 6.28).

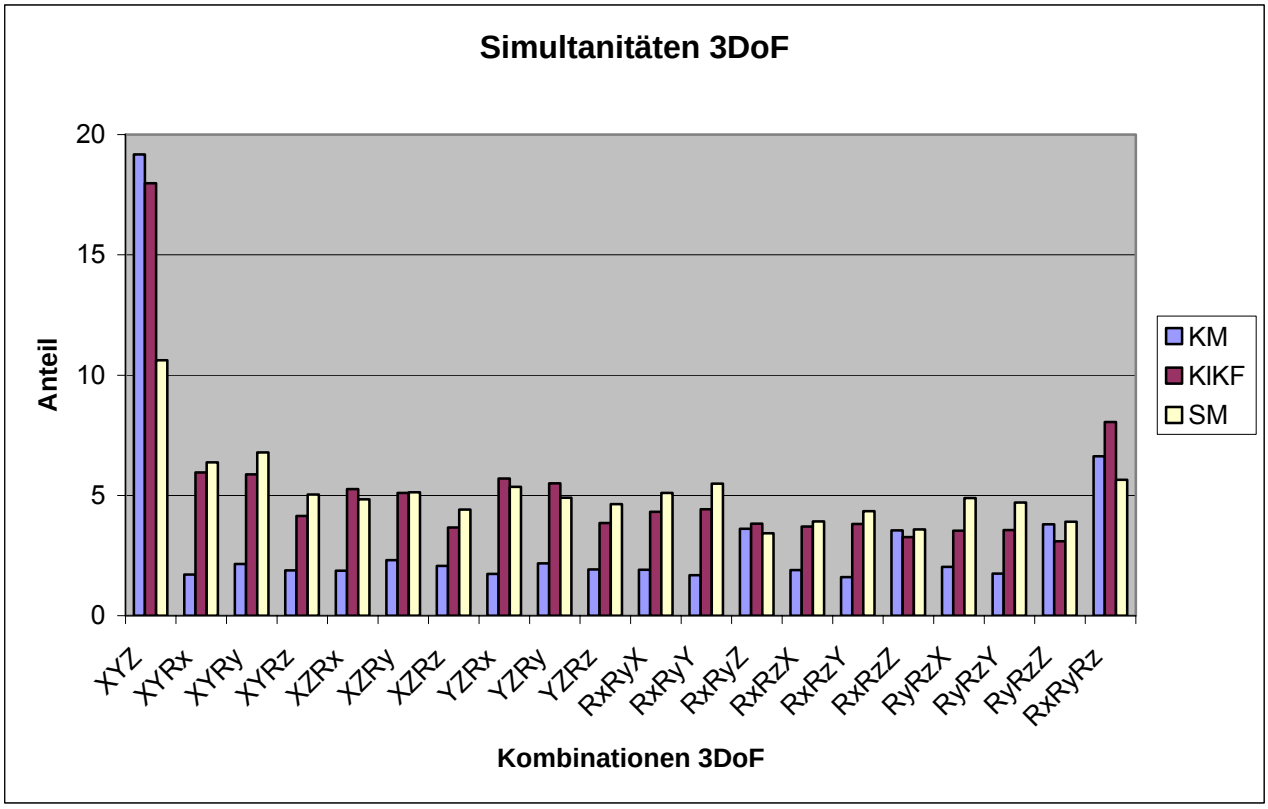

Abb. 6.28:

Hier treten ähnliche Effekte, wie im Abschnitt 5.2.4 Simultanitäten innerhalb 2DoF beschrieben, auf.

Bei Kugelmaus und kleinen Kugelfisch stellen die reine Translationskombinationen erneut den größten Anteil dar. Es folgen die reinen Rotationskombinationen. Die geringsten Anteile kommen von den Translations-Rotations-Kombinationen. Die Effizienz dieser Kombinationstypen unterscheidet sich erkennbar. Bei der SpaceMouse liegen die Anteile der verschiedenen Kombinationstypen wiederum deutlich näher beieinander, d.h. deren Effizienz ist auf einem ähnlichen Niveau.

6.2.6 Simultanitätslernverhalten der SpaceMouse:

Die beiden folgenden Grafiken zeigen die Entwicklung des Simultanitätsverhalten der SpaceMouse von Session1 zu Session4 (Abb. 6.29, 6.30), noch einmal im Detail.

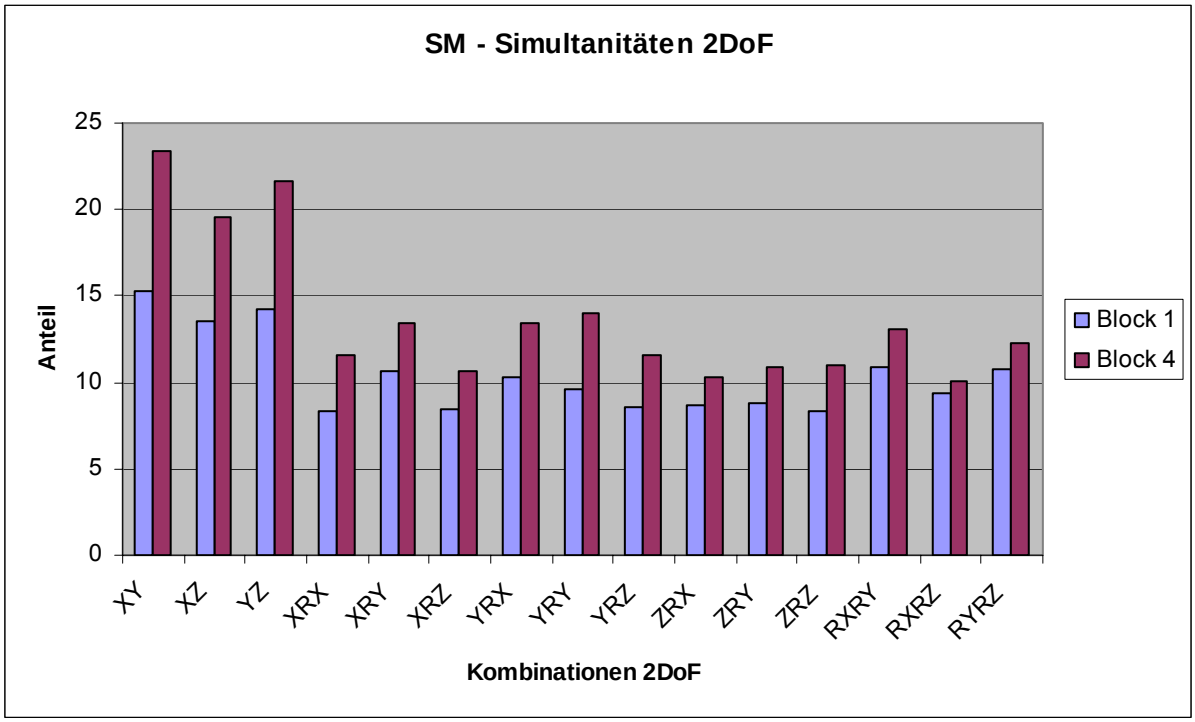

Abb. 6. 29:

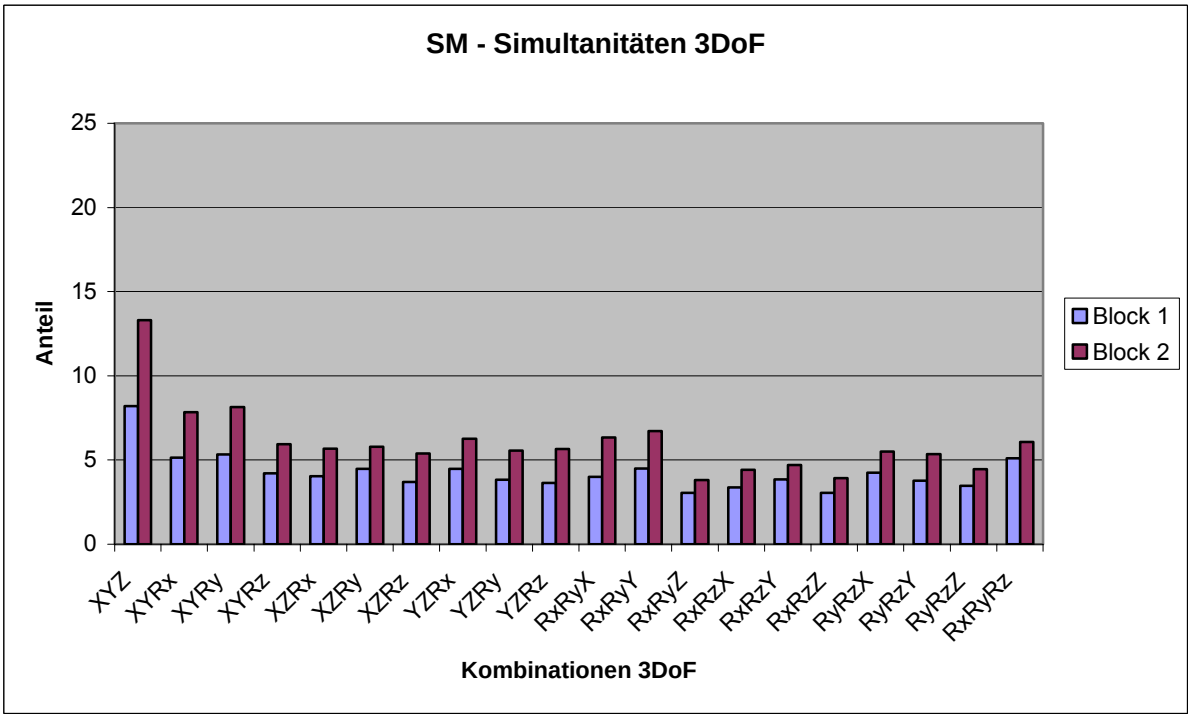

Abb. 6.30:

Sowohl bei 2 DoF als 3 DoF–Simultanitäten zeigen alle möglichen Kombinationen einen deutlich erhöhten Prozentanteil. Besonders bei den reinen Translationskombinationen ist der Zuwachs sehr groß.

Die zweite Hypothese besagte, dass bei der SpaceMouse Separierbarkeit gelernt werden würde. Offensichtlich ist dies nun hier nicht der Fall, da alle Freiheitsgradkombinationen über die 4 Sessions hinweg, beständig häufiger ausgeführt werden. Wir erweitern unsere Betrachtung auf alle 6 DoF

Die beiden folgenden Kreisdiagramme (Abb. 6.31, 6.32) zeigen die verschobenen Anteile von 0 bis 6 DoF bezüglich der mittleren Trialzeit, von Session 1 zu Session 4 hin.

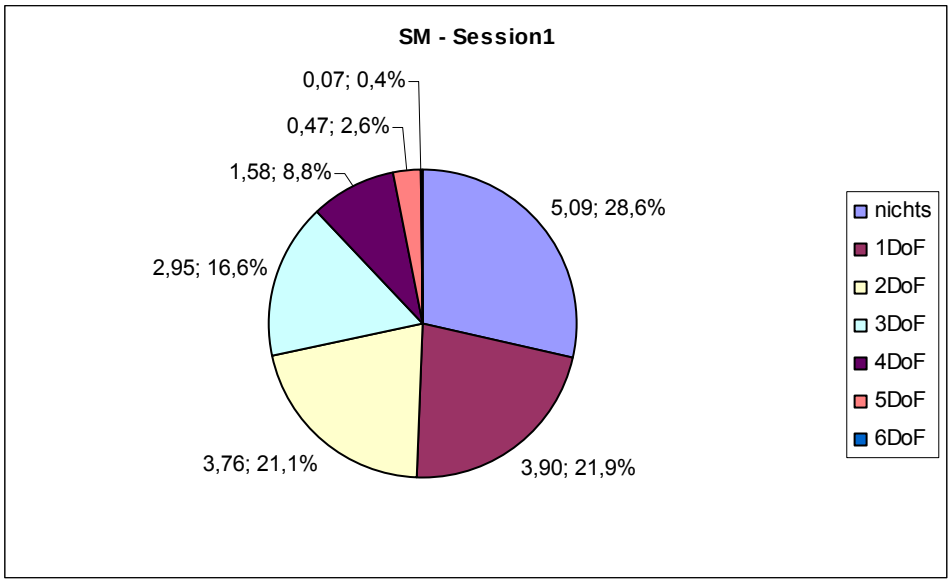

Abb. 6:31:

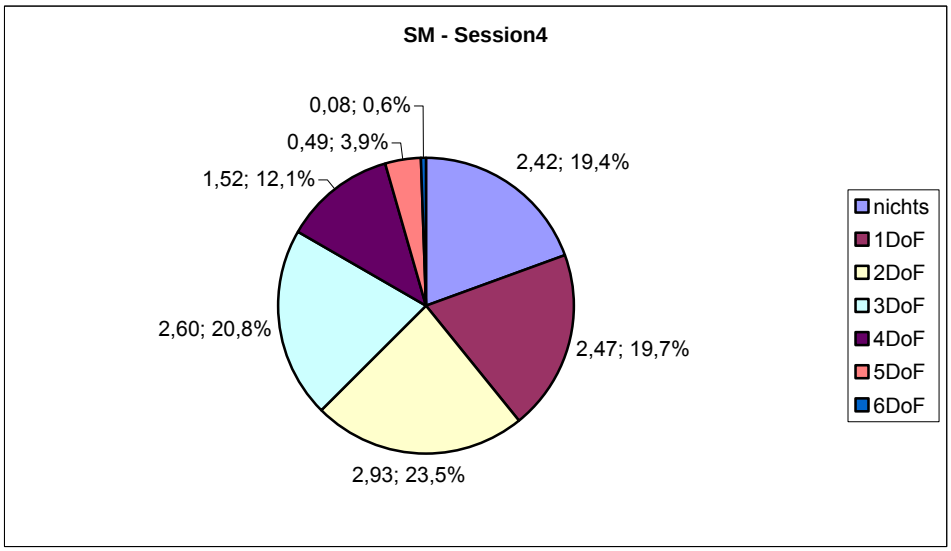

Abb. 6:32:

Auch hier ist wieder zu sehen, dass sich das Verhältnis von Einzelaktionen (1 DoF) zu simultanen Verhalten hin verschiebt. Der Anteil von Kombinationen ≥ 3 DoF liegt in Session 1 bei 28,4% und in Session 4 sogar bei 37,4%.

Es ist schwer zu glauben, dass gerade die hohen DoF–Kombinationen alle bewusst von den Versuchspersonen so geplant wurden und auch umgesetzt werden konnten. Daher stammte auch die These zur Separierbarkeit der SpaceMouse. Sie kann aber aufgrund der konsequent steigenden Simultanitätsanteile nicht mehr bei beibehalten werden.

Wir schließen daraus, dass die Versuchspersonen nicht weiter versuchen DOFs zu separieren, sondern versuchen mit den vielen Streuinputs auf den benachbarten Kanälen umzugehen. Dazu soll nun ein Trial der SpaceMouse im Detail angeschaut werden. In der Grafik (Abb. 6.33) sind die Inputs der 6 DOF über die Zeit angetragen.

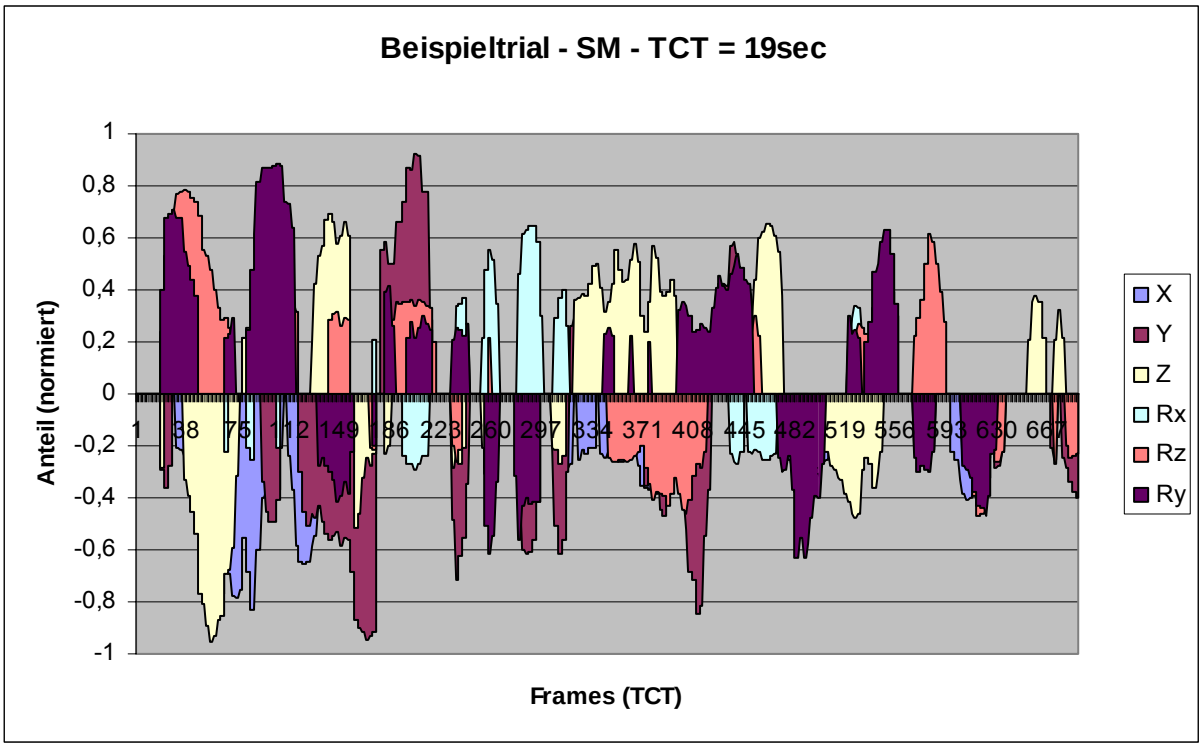

Abb. 6.33:

Auf vielen Kanälen, besonders bei RY, ist zu sehen, dass sehr viele Richtungswechsel aufeinander folgen. Vom positiven wird in den negativen Bereich

gewechselt bzw. umgekehrt. Dies scheint dafür zu sprechen, dass häufig übersteuert wird und es sehr schwer fällt die SpaceMouse wunschgemäß zu bedienen. Die Versuchspersonen sind daraufhin gezwungen, häufig Gegenbewegungen auszuführen.

Dies stützt erneut unsere Haupthypothese, dass Geschwindigkeitssteuerung der Rotation schwer zu kontrollieren ist.

6.2.7 Simultanitätslernverhalten der Kugelmaus

Die beiden folgenden Grafiken zeigen die ungewollten Simultanitätseffekte der Kugelmaus zwischen Translation und Rotation jeweils an 2DoF-Kombinationen (Abb. 6.34) und an 3DoF-Kombinationen (Abb. 6.35). Durch das Umgreifprinzip sollten diese Kombinationen eigentlich nicht vorhanden sein.

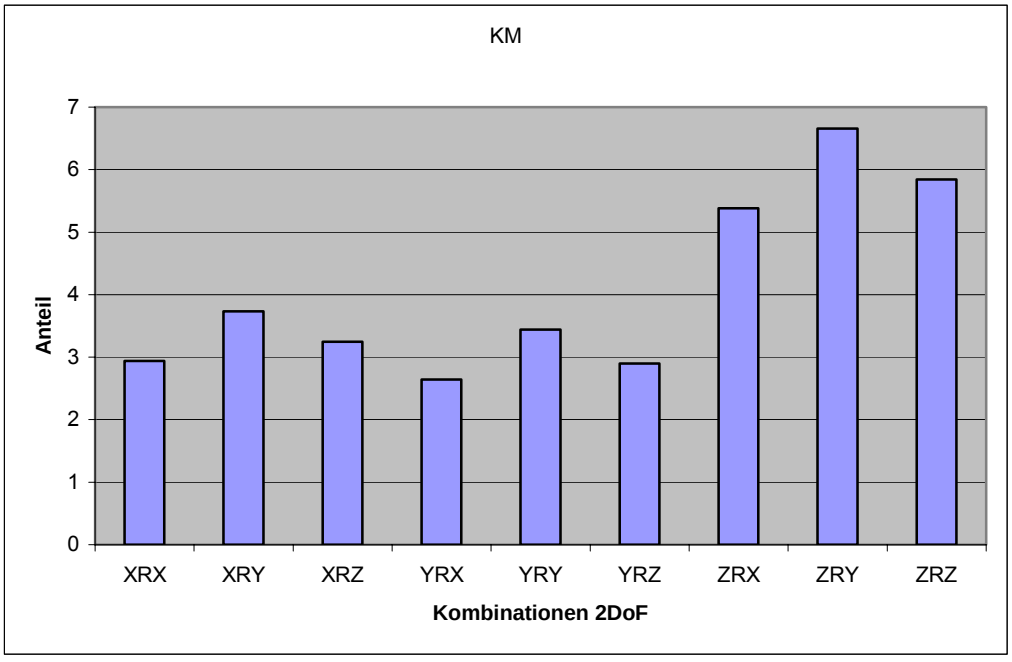

Abb. 6.34:

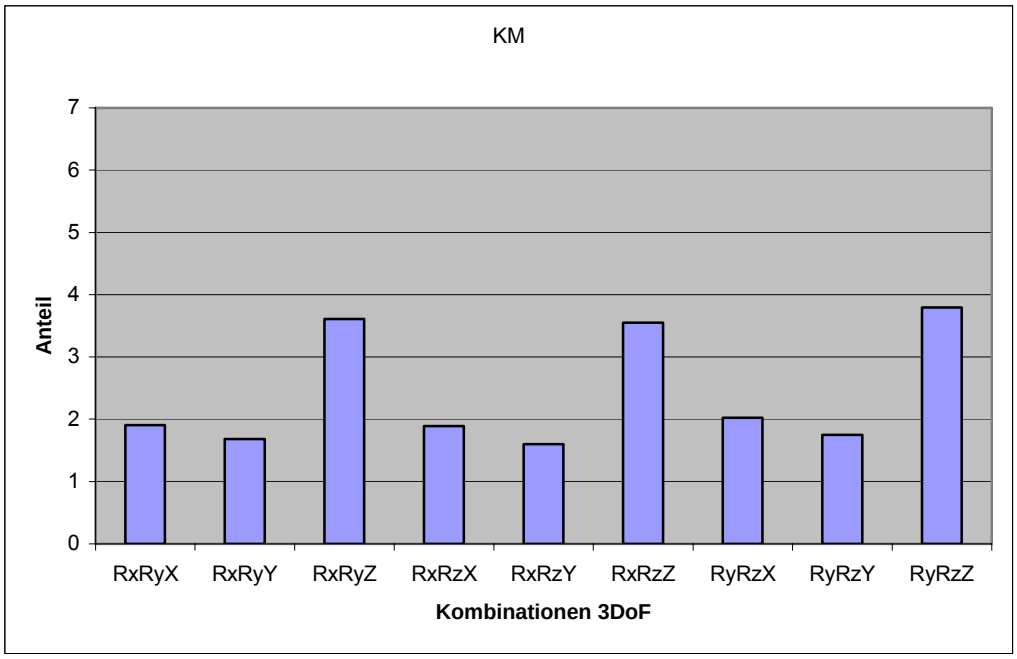

Abb. 6.35:

Wie in der Gerätevorstellung (Kapitel 2.5) erwähnt, ist bei der Kugelmaus die Kugel auf dem Gehäuse einer SpaceMouse angebracht. Beim Drehen der Kugel ist es möglich, dass unabsichtlich ein Input auf die Translationskanäle der eingebauten SpaceMouse wirkt. Insbesondere in Z-Richtung, wo das Gewicht der Finger die Kugel (und damit das gesamte Gehäuse) leicht nach unten drücken kann. Kombinationen mit einem Z-Anteil sind folglich höher als solche ohne Z-Anteil. Korrekterweise müssten die Anteile der eben gezeigten Kombinationen auf die jeweilige Einzelrotation oder Rotationskombinationen angerechnet werden.

Neben mechanischen Verbesserungen, kann auch von der Implementationsseite den ungewollten Simultanitäten begegnet werden. Unsere programmtechnische Umsetzung ist in dieser Hinsicht sicherlich nicht optimal. Als bessere Alternative könnten die Inputs entweder von Rotation oder Translation jeweils komplett ignoriert werden. Solange die Kugel manipuliert wird und Inputs generiert, werden die Translationskanäle gesperrt. Wird dagegen der Sockel verschoben, können die Rotationsinputs fallengelassen werden.

6.2.8 Simultanitätslernverhalten des kleinen Kugelfischs

Hier kann noch einmal die 1. Simultanitätshypothese bestätigt werden. Die Kreisdiagramme zeigen die Verschiebung der verschiedenen DoF-Anteile von Session 1 zu Session 4 (Abb. 6.36, 6.37).

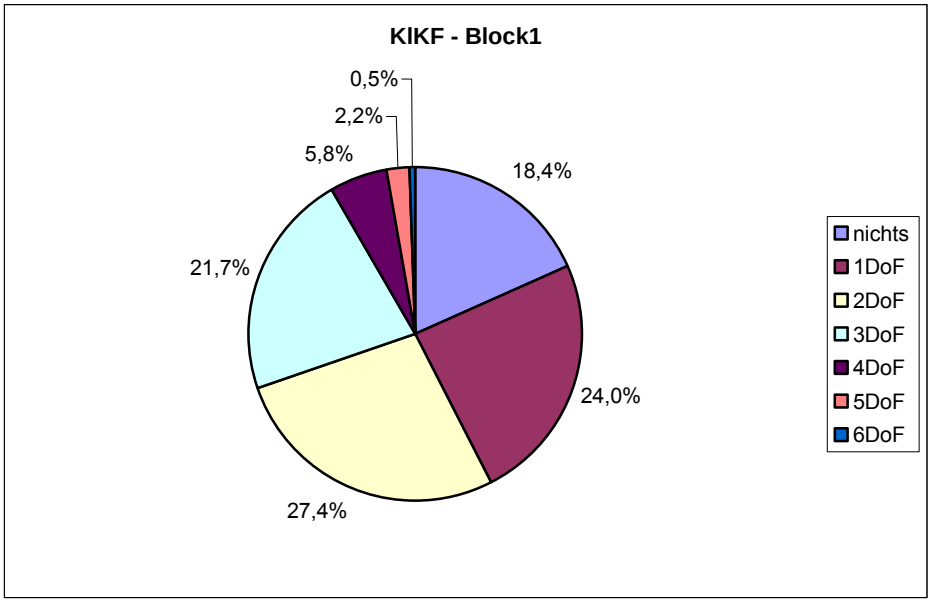

Abb. 6.36:

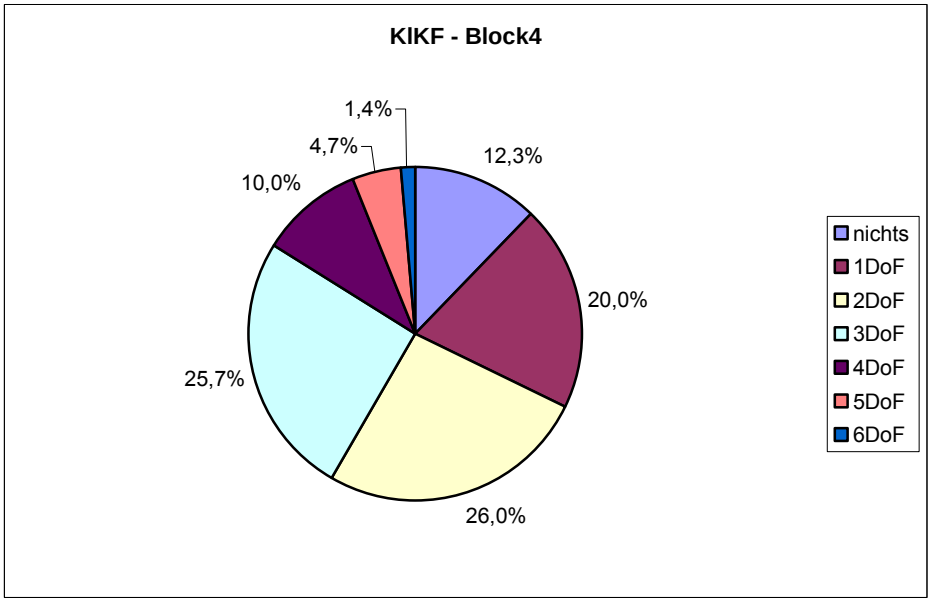

Abb. 6.37:

Die Verteilung verschiebt sich von Kombinationen ≤2 hin zu stärker simultanen Aktionen. Besonders die 3 und 4-DoF–Kombinationen werden verstärkt eingesetzt.

6.3 Fragebogenauswertung

Wie schon beim Versuchdesign erwähnt wurde, haben wir nicht nur die „harte“ Fakten (wie die TCTs) durch die Tests ermittelt, sondern auch der subjektive Eindruck der einzelnen Testpersonen ist von Interesse, um die Eingabegeräte zu bewerten. Zum Erfassen dieser Eindrücke kamen Fragebögen zum Einsatz. Den Aufbau und die Reihenfolge der Fragebögen ist bereits bei den Versuchsmodalitäten (Kapitel 4.6) besprochen wurden. An dieser Stelle soll es vielmehr um die Auswertung der Fragebögen gehen, wobei natürlich folgender Aspekt von Hauptinteresse ist:

Lassen sich Aussagen, die anhand der TCTs getroffen wurden durch die Ergebnisse der Fragebögen stützen?

Des Weitern sind natürlich auch die auf den Fragebögen hinterlassenen Kommentare der Versuchpersonen zu den einzelnen Geräten äußerst aufschlussreich.

Die Angaben der Versuchspersonen wurden für die drei Geräte, SpaceMouse, kleiner Kugelfisch und Kugelmaus, unter Verwendung des Allgemeinen Linearen Modells (SPSS), auf Signifikanz getestet.

Zuerst einmal liefert uns der Fragebogen Aufschluss über die Spannweiter der Parameter (Alter, Geschlecht, Handlänge etc.) der Versuchspersonen. Die Stichprobe setze sich aus 8 weiblichen und 8 männlichen Probanden zusammen. Das Schnitt waren die Versuchspersonen 23 Jahre alt, hatten eine mittlere Handlänge (Handwurzel bis Spitze Mittelfinger) von 18,78 cm, eine mittlere Handspannweite von 19,25 cm und das getestete Stereosehvermögen pegelte sich im mittel bei 12,37 cm ein (Abb. 6.38).

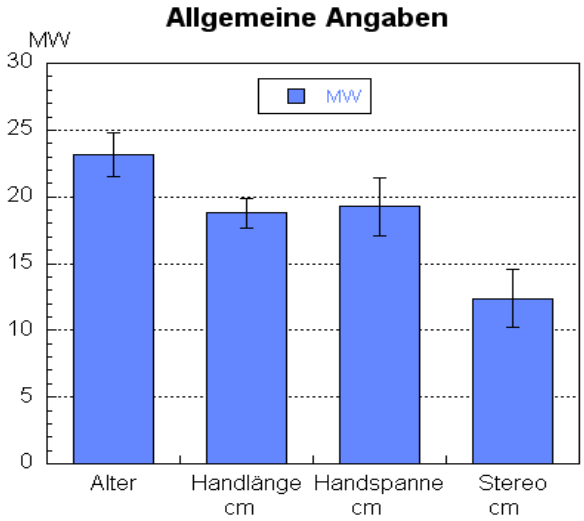

Abb. 6.38:. Allgemeine Parameter der Probanden

Wie man der Abbildung entnehmen kann handelte es sich um eine sehr homogene Stichprobe, was sich in den Standartabweichungen wiederspiegelt. Daraus lässt sich schlussfolgern, dass etwaige Ungereimtheiten der Daten nicht auf die Stichprobe zurückzuführen sind sondern in etwas anderem begründet sein müssen.

Die Geschlossenheit der Stichprobe setzt sich bei anderen erfragten Parametern fort. So zum Beispiel haben alle Probanden Computererfahrung, 15 von 16 haben schon mit anderen Eingabegeräten, außer der Mause gearbeitet (Touchpad, Joystick, Gamepad etc.) (Abb. 6.39) und 14 von ihnen hatte auch schon Berührungen mit 3D-Welten sei es nur beim Spielen oder beim Arbeiten (Abb. 6.40).

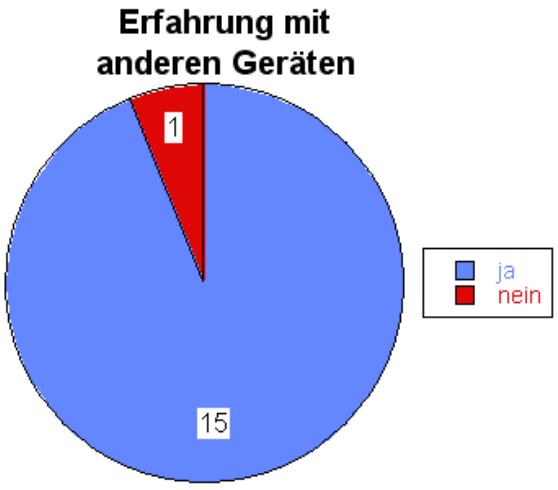

Abb. 6.39: Computererfahrung der Probanden.

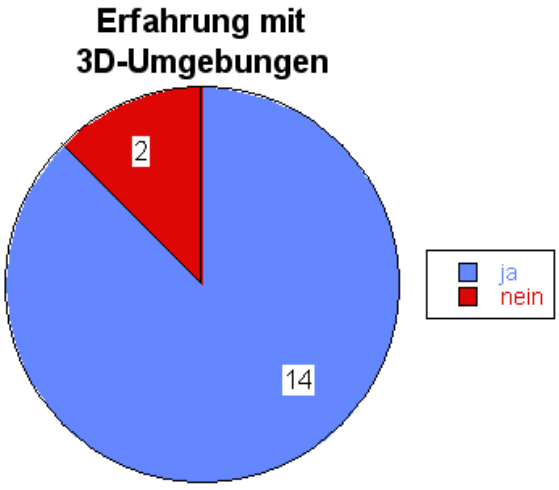

Abb. 6.40: 3D-Erfahrung der Probanden.

Diese Tatsachen lassen darauf schließen, dass keiner der Probanden so überrascht bzw. fasziniert von der Aufgabe gewesen sein wird, dass es dadurch zu einer Verfälschung der Daten gekommen wäre. Dies lässt sich auch durch die Beobachtung der Testleiter stützen. Alle Probanden waren sich der Testsituation bewusst und haben konzentriert an der Aufgabe gearbeitet. Wohingegen während des Rundgangs beobachtet werden konnte, dass häufig Besucher so sehr von 3D-Darstellung fasziniert waren, dass sie überhaupt nicht die Intention der Testreihe wahrnahmen. Dies kann jedoch bei den Probanden ausgeschlossen werden.

Zunächst gilt es zu untersuchen, ob die beiden Sieger der TCT-Auswertung „kleiner Kugelfisch“ und „Kugelmaus“ auch auf den Fragebögen und somit in der Gunst der Probanden vorne liegen. Um diese Frage zu klären schauen wir uns die allgemeine Bewertung der Probanden an. Wir baten sie nachdem sie mit allen vier Geräten die Testreihen absolviert hatten, ihren Favoriten und ihren Verlierer des Quartetts zu nennen. Es ergab sich folgendes Bild (Abb. 6.41, 6.42).

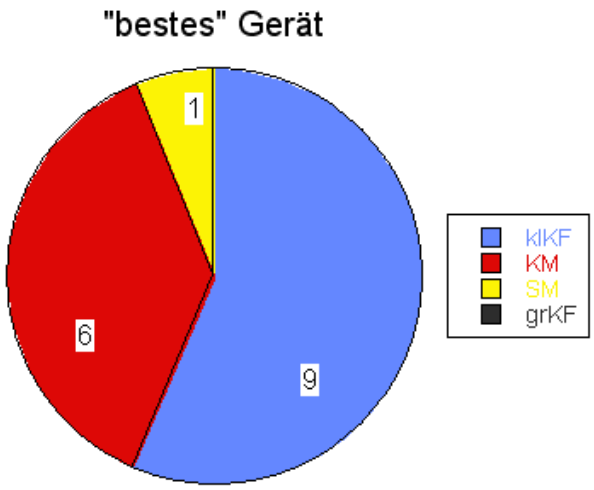

Abb.6.41: Nennungen „bestes“ Gerät

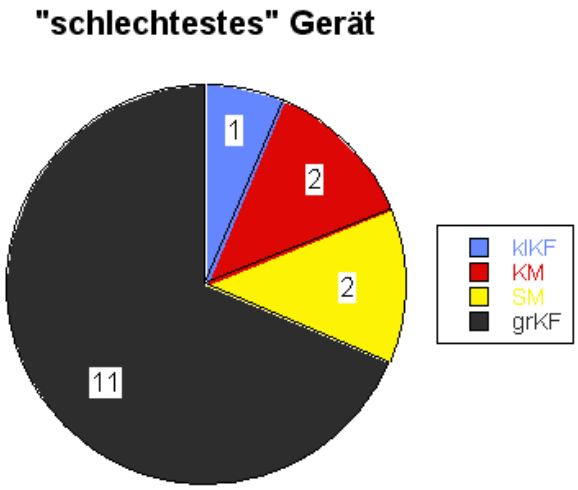

Abb. 6.42: Nennungen „schlechtestes“ Gerät

Wie man den Grafiken entnehmen kann liegen der kleine Kugelfisch und die Kugelmaus auch in der subjektiven Bewertung der Versuchpersonen klar vor SpaceMouse und großem Kugelfisch. Dementsprechend stellt sich das Ergebnis der Frage nach dem schlechtesten Gerät dar. Diese Statistik wird klar vom großen Kugelfisch angeführt mit 11 Nennungen. Die anderen Geräte präsentieren sich mit etwa gleichwenigen Nennungen. Diese subjektiven Präferenzen spiegeln eindeutig die Platzierungen der Geräte anhand der erreichten TCTs wieder. Das schlechte

Abschneiden des großen Kugelfisches, sowohl was die TCTs als auch die Probandenmeinung angeht ist sicher auf die mechanischen Unzulänglichkeiten zurück zu führen. Um das Bild das uns diese beiden Kreisdiagramme bieten zu untermauern seine an dieser Stelle noch zwei Resultate der Fragebogenauswertung angeführt. In Abbildung 6.43 veranschaulicht die Antworten der Probanden auf die Frage: *„Wie war das Gerät für sie zu bedienen?“*

Die Angaben der Versuchspersonen über die Bedienbarkeit waren signifikant für die drei Geräte (F = 9.617, p= 0.001). Der große Kugelfisch wurde bei sämtlichen, folgenden Signifikanzbetrachtungen nicht mit berücksichtigt.

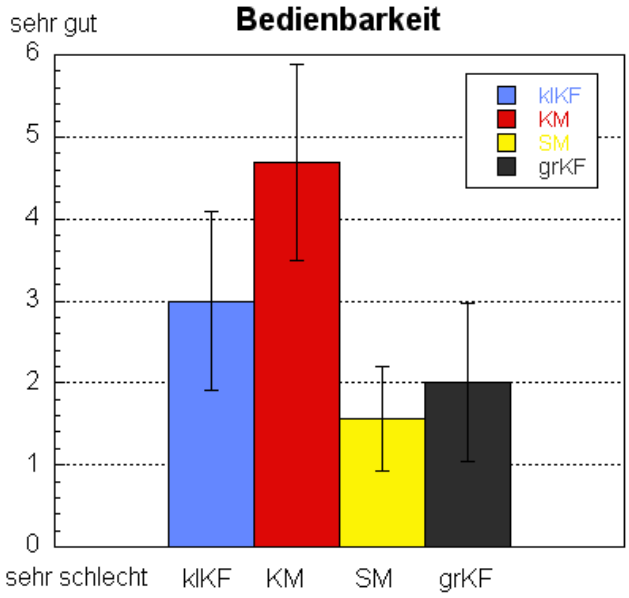

Abb. 6:43: Grafik zu Frage nach Bedienbarkeit der einzelnen Geräte

In dieser Frage fließen natürlich sämtliche Parameter (hardware- und softwareseitig) der Geräte in die Bewertung mit ein. Somit kann man hier noch keine Aussagen über konkrete Vor- bzw. Nachteile eines Gerätes ableiten, sondern es ist vielmehr ein Gesamteindruck der herausgestellt wird.

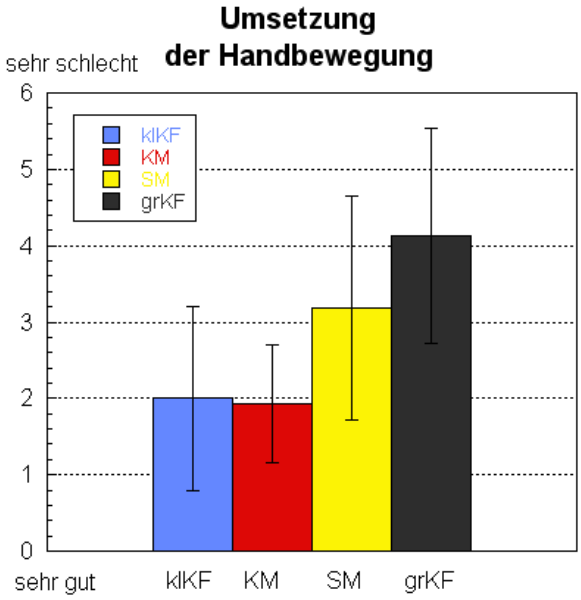

Abb. 6.44: Grafik zu Frage nach Umsetzung der Handbewegung auf Cursorsteuerung

Selbiges gilt auch für die in Abbildung 6.44 dargestellten Resultate auf die Frage „Wie empfanden sich die Umsetzung der Handbewegung auf die Cursorsteuerung?“ Diese Bewertung war ebenfalls signifikant: $F = 5{,}105$, $p < 0.02$.

Beide Grafiken (Bedienbarkeit, Umsetzung der Handbewegung) unterstützen die Ergebnisse der TCT-Auswertung und die subjektiven Präferenzen der Probanden. Den Grafiken ist zu entnehmen das sich der kleine Kugelfisch und die Kugelmaus auf etwa gleichem Niveau bewegen und das Testfeld anführen. Weit abgeschlagen auch hier wieder große Kugelfisch. Die SpaceMouse ist zwischen beiden Extrema angesiedelt.

Um nun herauszufinden, warum sich die Platzierungen der Geräte so darstellen, schauen wir die Fragen an, bei denen es um spezielle Eigenschaften bzw. Parameter der einzelnen Geräte geht. Allerdings immer auch mit den zu Beginn der Studie aufgestellten Hypothesen im Hinterkopf.

Wenn man sich die oben angeführten Grafiken anschaut, fällt natürlich sofort das schlechte Abschneiden des großen Kugelfisches ins Auge. Folglich sollen hier zuerst Gründe dafür anhand der Fragebögen aufgezeigt werden. Uns selbst sind die Gründe dafür bewusst. Die unzulängliche Mechanik schränkte dieses Gerät doch sehr ein (Kapitel 2.3). Dies schlug sich selbstverständlich in den Antworten der

Probanden nieder. So wurde explizit nachgefragt wie einfach sowohl Translation als auch Rotation mit dem entsprechenden Gerät zu bewerkstelligen seien. Schaut man sich die Resultate des großen Kugelfisches bei der Frage: *„Wie leicht fiel es ihnen den Cursor zu verschieben?“* an (Abb. 6.45), sieht man deutlich die Schwäche des Gerätes bei der Translation. Die Angaben zur Translation waren ebenfalls signifikant (F=7.959, p=0.004).

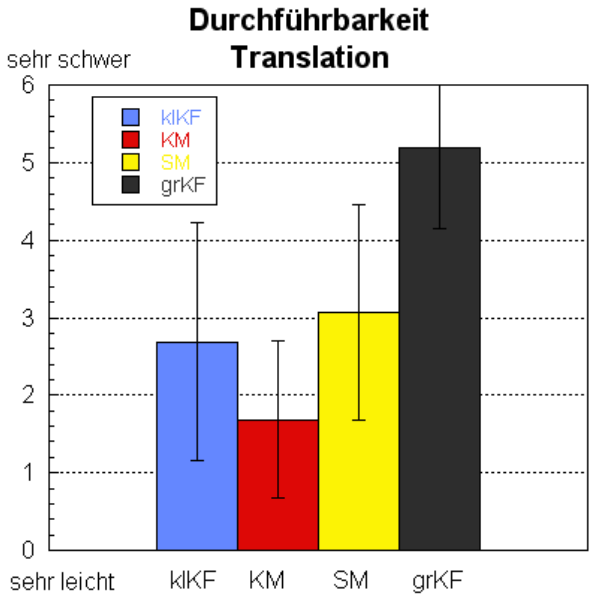

Abb. 6.45: Durchführbarkeit der Translation

Der große Kugelfisch wurde deutlich schlechter bewertet, als alle anderen drei Geräte. Das die Translation mit dem großen Kugelfisch so schwer fiel, liegt in der verwendeten Technik begründet (Potentiometer). Die Führung der Potentiometer hatte einen gewissen Reibungswiderstand und durch zu schwache Rückzugsfedern keine eindeutige Nullposition. Diese beiden Umstände sind dafür verantwortlich, dass zum einen die Bedienung mit einem hohen Kraftaufwand verbunden ist und zum anderen öfters korrigierendes Steuern notwendig war. Daraus resultieren stärkere Ermüdungserscheinungen der Handmotorik, als es bei allen anderen Geräten der Fall ist (Abb. 6.46). Die unterschiedlichen Bewertungen für Ermüdungen sind als sehr signifikant zu bezeichnen: F = 13,555, p= 0.001.

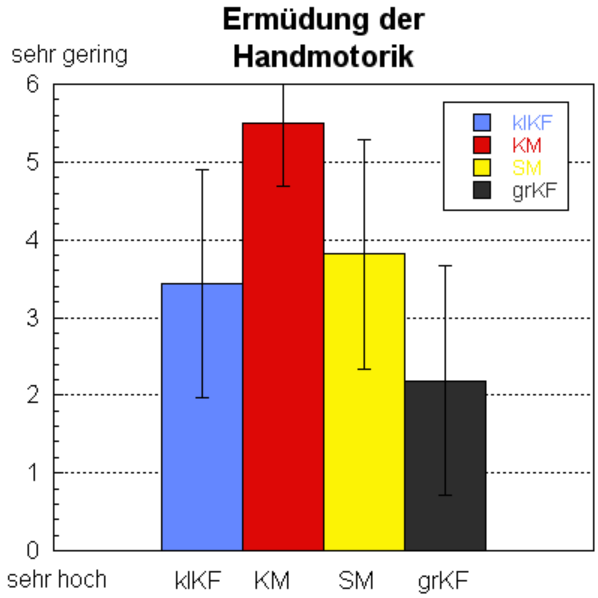

Abb. 6.46: Ermüdungserscheinungen der Handmotorik.

Die Abbildung zeigt darüber hinaus auch ein sehr gutes Abschneiden der Kugelmaus, was die Ermüdungserscheinungen der Handmotorik angeht. Sie ist nicht nur im Schnitt als bestes Gerät im Bezug auf die Ermüdungserscheinungen aus dem Test hervorgegangen sondern sie weißt auch die geringste Streuung in den Bewertungen auf. Grade bei Desktopeingabegeräten ist dies ein durchaus wichtiges Kriterium, da ein potentieller Anwender durchaus einen ganzen Arbeitstag mit dem Gerät zubringen könnte. Es sei angemerkt, dass die Testpersonen im Schnitt ca. nur 30 Minuten das Eingabegerät bedienten und sich trotzdem schon solche Unterschiede herauskristallisierten.

Kommen wir nun zu wesentlichen Aspekt in dem sich die an der BUW entwickelten Geräte von der kommerziellen Spacemouse unterscheiden: die Kugel zur isometrischen Rotationssteuerung. Tatsächlich zeigt die Grafik, dass die Rotationssteuerung mittels Kugel von den Probanden bevorzugt wurde (Abb. 6.47). Leider sind die Angaben der Rotation jedoch nicht signifikant ($F=2.752$, $p=0.090$), sie weisen jedoch eine Tendenz auf. Es könnte jedoch sein, dass sich die Kugelgeräte signifikant von der SpaceMouse unterscheiden. Der große Kugelfisch sei auch bei dieser Betrachtung ausgeklammert. Es ist interessant zu sehen das auch beim Aspekt der Rotationssteuerung der große Kugelfisch schlecht abschneidet, da er schließlich auch das Kugelelement für die Manipulation beinhaltet, wie auch die

besser bewerteten Kugelmaus und der kleine Kugelfisch. Woran könnte dies liegen? Eine mögliche Erklärung könnte sein, dass sich die negativen Eigenschaften des großen Kugelfisches beim Translationsverhalten im Gesamtbild des Gerätes niederschlagen. Gemeint ist damit der subjektive Gesamteindruck der Testpersonen vom Eingabegerät. Kann der Proband aufgrund einer speziellen Unzulänglichkeit des Gerätes die gestellte Aufgabe nicht sonderlich gut erfüllen, so wird der Proband frustriert, und die Einschätzung des Gerätes sinkt.

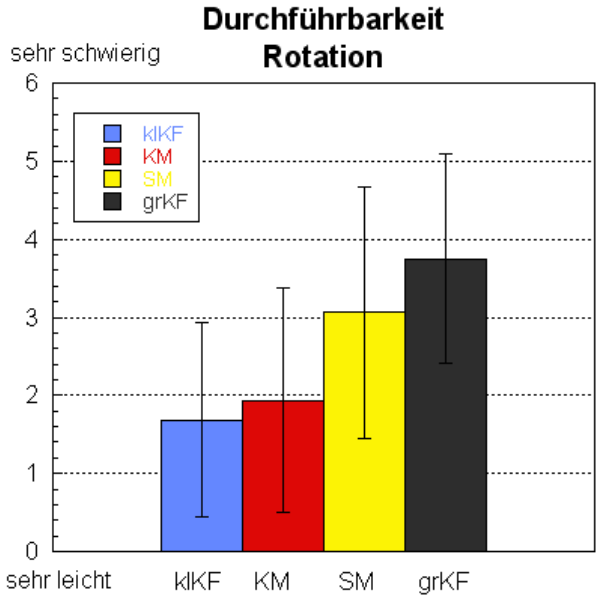

Abb. 6.47: Durchführbarkeit der Rotation

Und so schlägt sich dann diese negative Meinung über das Gerät auch in Fragen nach speziellen anderen Aspekten des Gerätes nieder. Es könnte also eine Art Objektivitätsverlust sein, der die Probanden zu der oben visualisierten Bewertung bewogen hat. Ein andere Möglichkeit wäre, das die größere Kugel des großen Kugelfisches führ diese Bewertung verantwortlich ist, allerdings scheint das nicht der Fall zu sein wie die nächste Grafik (Abb. 6.48) zeigt.

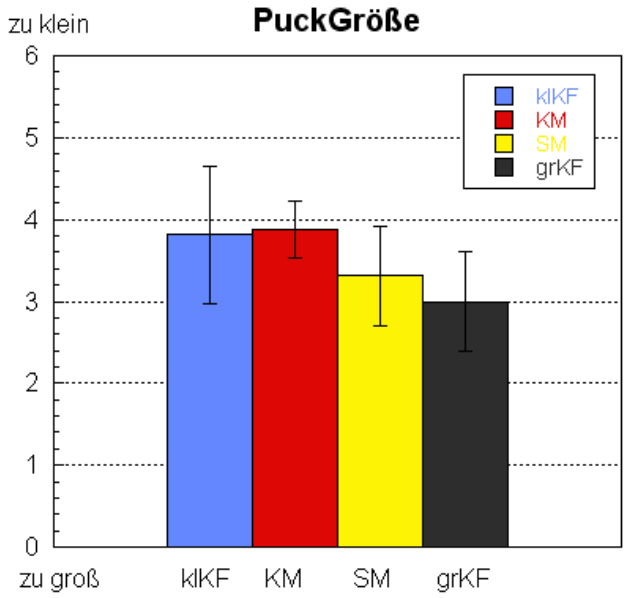

Abb. 6:48: Aussagen über die PuckGröße.

Erstaunlicherweise wird die Größe der Kugel beim großen Kugelfisch sogar als optimaler angesehen als es bei den beiden Geräten mit kleinerer Kugel der Fall ist. Sehr interessant ist auch die Tatsache, dass Kugelgröße bei Kugelmaus und kleinem Kugelfisch im Mittel gleich bewertet wird, jedoch sich die Standardabweichung deutlich unterscheiden. Dies kann nur auf die Unterschiede beim restlichen Aufbau der beiden Geräte zurück zuführen sein. Dadurch sind auch die Handhaltung bei der Bedienungen zwei verschiedene. Vielleicht hängt also die optimale Kugelgröße von der Handhaltung bei der Bedienung ab. Ob dies so ist könnten weiterführende Studien in diesem Bereich zeigen. Leider unterscheiden sich die Bewertungen der Puck- bzw. Kugelgröße nicht signifikant ($F=3.171$, $p=0.076$) voneinander. Daher kann die Nebenhyphothese-2 zumindest aus subjektiver Sicht der Probanden nicht bestätigt werden.

Die Einbindung der Geräte in die Testanwendung hat ebenfalls Einfluss auf die Bewertung der einzelnen Geräte. Hiermit sind ist die programminterne Verarbeitung der Geräteinputs gemeint. Aufgrund der baulichen Unterschiede der Geräte war keine universelle Lösung möglich. Um trotzdem Vergleichbarkeit zu gewährleisten waren wir bemüht die Softwareeinstellungen so zugestalten, dass kein Gerät bevor- oder benachteiligt wird. Das uns dies im Großen und Ganzen gelungen ist, zeigt die folgende Grafik (Abb. 6.49).

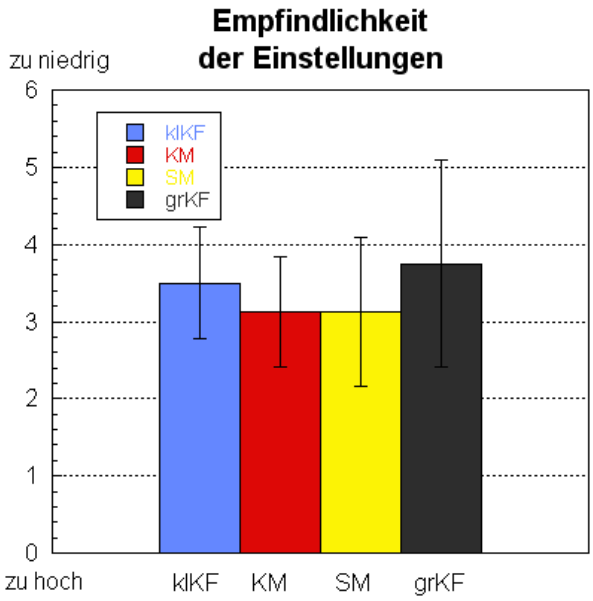

Abb. 6.49:

Keines der Geräte sticht hier nach Ansicht der Probanden hervor was die Empfindlichkeit der Einstellung betrifft. Das einzige auffällige ist, dass der große Kugelfisch eine größere Standardabweichung aufweißt, als die anderen Geräte. Auch diesmal lässt sich dies Ergebnis sicher auf die mechanischen Unzulänglichkeiten des großen Kugelfisches zurückführen.

Abschließend zur subjektiven Betrachtung sind an dieser Stelle noch die Bemerkungen der Probanden in tabellarischer Form zusammengefasst:

	SM	KM	klKF	grKF
Steuerung Rotation	• wenig intuitiv • kann schlecht isoliert ausgeführt werden • wird beeinflusst von Translation • beeinflusst Translation	• intuitiv • einfach/leicht • sehr sensibel • nur separiert ausführbar	• intuitiv • einfach/leicht • sensibel • kann sowohl separat, als auch in Kombination mit Translation ausgeführt werden	• intuitiv • einfach/leicht • sensibel • kann sowohl separat, als auch in Kombination mit Translation ausgeführt werden
Steuerung Translation	• einfach/leicht • wird beeinflusst von Rotation • beeinflusst Rotation	• einfach/leicht • getrennt von Rotation	• schwergängig • kann sowohl separat, als auch in Kombination mit Rotation ausgeführt werden	• sehr schwergängig • kann sowohl separat, als auch in Kombination mit Rotation ausgeführt werden
Handlichkeit/ Ermüdungserscheinungen	• handlich • gewohnte & entspannte Handhaltung • selten Ermüdungserscheinungen	• handlich • gewohnte & entspannte Handhaltung • selten Ermüdungserscheinungen • Umgreifen nötig	• unhandlich • unnatürlich/ evtl. unangenehme Handhaltung • häufig Ermüdungserscheinungen	• unhandlich • unnatürlich/ evtl. unangenehme Handhaltung • häufig Ermüdungserscheinungen
Sonstiges	• Leichter Sockel → hebt manchmal ab	• Kugel sitzt nur lose in der Fassung → springt manchmal heraus	• standfest	• standfest

7. Konverter

7.1 Motivation

Durch den Test wurden die Geräte evaluiert, und es zeigte sich, dass die an der BUW entwickelten Geräte konkurrenzfähig zum kommerziellen Produkt der SpaceMouse der Firma Logitech sind. Allerdings ist der Vergleich der Geräte notwendigerweise durch eine künstlich geschaffen Testaufgabe vollzogen worden. Nicht mehr, aber auch nicht weniger. Nun wäre es aber sicher interessant, ob sich die dadurch erzielten Ergebnisse in der Praxis bestätigen lassen. Dafür wäre es wünschenswert mit den an der BUW entwickelten Geräten Anwendungen steuern zu können, die üblicherweise mit der SpaceMouse bedient werden. Die Frage die sich stellt ist, wie reagieren z.B. Konstrukteure, wenn sich plötzlich ihre gewohnte CAD-Anwendung mit einem unserer Geräte steuern könnten? Nun gab es die Option einen Treiber für das einzelne Gerät zu schreiben, was bedeuten würde, dass man sich auch mit einer Schnittstelle vom Treiber zur betreffenden CAD-Anwendung kümmern müsste. Ebenso müsste dies für andere Anwendung (z.B. Maya, 3Dsmax etc.) vollzogen werden. Deshalb entschieden wir uns dafür, die bestehenden Treiber- und Schnittstelleninfrastruktur der SpaceMouse zu benutzen. Dies bedeutet, dass die Inputs unseres Eingabegerätes abgefangen werden, auf das SpaceMouse-Protokoll angepasst / konvertiert und anschließend mittels Nullmodemkabel an die serielle Schnittstelle des Zielrechners geschickt werden. Auf diesem Zielrechner läuft die entsprechende Anwendung die man steuern möchte, ebenso wie der SpaceMouse-Treiber von 3DConnexions. Der Treiber hört nun die serielle Schnittstelle ab und handelt als ob dort eine SpaceMouse angeschlossen wäre.

Dies wurde für den Testsieger realisiert. Im weiteren Abschnitt wird genauer auf dieses Konvertierung eingegangen.

7.2 Das SpaceMouse-Protokoll

Um dem SpaceMouse-Treiber eine SpaceMouse an der seriellen Schnittstelle vorgaukeln zu können, war es nötig sich mit dem SpaceMouse-Protokoll auseinander zu setzen, um so zu wissen wie die Inputs konvertiert werden müssen.

Glücklicherweise stellt die Firma 3DConnexions ein entsprechendes Dokument zur Verfügung dem die gewünschten Informationen entnommen wurden. Die folgenden Ausführungen beziehen sich auf diese Dokument (3D Motion Control Serial Protocol). Das SpaceMouse-Protokoll codiert die Bewegungsinformation in 26 Zeichen, Nippel genannt. Die Wertigkeiten der Nippel können der „Nippel-coding-table“ entnommen werden (Abb. 7.1).

Das erste Zeichen ist immer ein „d“ um den Beginn eines Datenpaket und als letztes Zeichen ein „/r“ um das Ende das Datenpaketes zu signalisieren. Dazwischen befinden sich die restlichen 24 Zeichen, die sich wie folgt zusammensetzen: 4 Zeichen (Nippel) codieren jeweils eine Manipulationsform. Beispielhaft sei hier folgende Zeichenkette angeführt:“dHBA5G?HKH000H0A6GNA6H06B/r“! Zuerst stehen die Translationen (rötlich hervorgehoben) entlang der Achsen (x,y,z), dann die Rotationen (bläulich hervorgehoben) um die Achsen (x,y,z). Um die übertragene Zeichenkette zu decodieren kommt folgende Formel zur Anwendung, für jedes Quadtupel: Wert = 4096 * 1. Nippel + 256 * 2. Nippel + 16 * 3. Nippel + 4. Nippel – 32768. Für die oben stehende Beispiel Zeichenkette ergibt dies also für *X = 4096 * 8 + 265 * 2 + 16 * 1 + 5 – 32768 = 533*. Es handelt sich also um eine Hexadezimale Codierung mit einem theoretischen Wertebereich von -32768 bis 32768 (16^4=65536, 65536/2= 32768). Aus der Umrechnung wird ersichtlich, dass positive Zahlen durch ein „H“ als erstes Nippel und negative Zahlen durch ein „G“ als ersten Nippel im Quadtupel der entsprechenden Manipulation entlang bzw. um die Achse dargestellt werden. Es ist also notwendig den Inputs des Eingabegerätes auf dieses Protokoll anzupassen.

Nibble Code	4 Bits	Character	Character Hexadecimal
0	0000	0	30H
1	0001	A	41H
2	0010	B	42H
3	0011	3	33H
4	0100	D	44H
5	0101	5	35H
6	0110	6	36H
7	0111	G	47H
8	1000	H	48H
9	1001	9	39H
A	1010	:	3AH
B	1011	K	4BH
C	1100	<	3CH
D	1101	M	4DH
E	1110	N	4EH
F	1111	?	3FH

Abb. 7.1: Nipple-Coding-Table

Input	Zeichen	Dezimal	berechnete Wert
X	HBA5	8,2,1,5	533
Y	G?HK	7,15,8,11	-117
Z	H000	8,0,0,0	0
A	H0A6	8,0,1,6	22
B	GNA6	7,14,1,6	-490
C	H06B	8,0,6,2	98

Abb. 7.2: Beispielnipplestring aufgeschlüsselt

7.3 Implementation

Um diese Aufgabe zu realisieren wurde die Programmiersprache C gewählt, da dort schon eine gewisse Grundfunktionalität zum Einlesen des Inputs vorhanden war. Des weiteren bietet C komfortable Möglichkeiten, um mit der seriellen Schnittstelle (RS232) umzugehen.

Es entstanden zwei Programme, um die Aufgabe zu lösen: ein Dämonprogramm und das Hauptprogramm. Das Dämonprogramm ist dafür verantwortlich die vorbereitende Kommunikation zwischen SpaceMouse-Treiber und Gerät abzuwickeln. Hierbei werden üblicherweise vor der Benutzung Treiber und SpaceMouse synchronisiert, sowie einige Konfigurationen (z.B. Steuerungsmodi, Blink- und Beepkommandos) vom Treiber an die SpaceMouse übermittelt.

Der grundlegende Aufbau des Hauptprogramms ergibt sich aus der Aufgabenstellung und stellt sich wie folgt dar:

Es werden alle entsprechenden Ports (USB und Seriell) initialisiert und für die entsprechende Benutzung (nur Lesen, nur Schreiben, Parametereinstellungen) vorbeireitet. Hierbei ist anzumerken, dass in C diese Schnittstelle nur als spezielle Datei angesehen und entsprechend behandelt wird.

Die nachfolgenden Aktionen laufen in einer Endlosschleife ab, da sie ständig wiederholt werden müssen.

Es wird Input vom Eingabegerät gelesen. Dieser Input wird dann auf einen gewissen Wertebereich, den die SpaceMouse üblicherweise liefert gemappt und anschließend in Hexadezimalform umgerechnet. Die so ermittelten Werte werden in der korrekten Reihenfolge (des SpaceMouse-Protokolls) an die serielle Schnittstelle geschickt. Von dort werden sie per Nullmodemkabel an den Zielrechner weitergeleitet.

Abschließend werden die initialisierten Ports wieder geschlossen.

Im Folgenden sollen das Vorgehen anhand von Quelltext aufgezeigt werden (Kommentare im Quelltext sind fett hervorgehoben).

Das Dämonprogramm erfüllt seine Aufgabe, indem es die vom Treiber geschickten

Kommandos wieder an den Treiber zurückgibt. Dies ist die Reaktion die 3DConnexions für die SpaceMouse vorgibt.

Zuallererst müssen natürlich verschiedene Bibliotheken eingebunden werden. Hier sind es folgende:

```
        #include <sys/types.h>
        #include <sys/stat.h>
        #include <fcntl.h>
        #include <termios.h>
        #include <stdio.h>
        #include <inttypes.h>
```

Dann erfolgen einige Makrodefinitionen:

```
        /* baudrate settings are defined in <asm/termbits.h>, which is
        included by <termios.h> */
        #define BAUDRATE B9600
        /* change this definition for the correct serial port */
        #define MODEMDEVICE "/dev/ttyS1"
        #define _POSIX_SOURCE 1 /* POSIX compliant source */

        #define FALSE 0
        #define TRUE 1
```

Anschließend werden noch Variablen und Konstanten festgelegt:

```
        int fd,c,send;
        int i = 0;
        int res =-1;
        int beepcount = 0;
        struct termios oldtio,newtio;
        /*varies response strings for driverrequests*/
  char buf[10];
        char beep[] = {'b','\r'};
        char status_mode[] = {'m','3','\r'};
        char compress_mode[] = {'c','3','G','\r'};
        char led_mode[] = {'l','0','0','0','\r'};
        char sensetiv_mode[] = {'q','0','0','\r'};
        char datarate_mode[] = {'p','A','A','\r'};
        char nullradius_mode[] = {'n','?','\r'};
        char version[]={'v',' ',' ','M','A','G','E','L','L','A','N',' ','
',' ','V','e','r','s','i','o','n',' ','5','.','7','9',' ',' ','b','y',' ',
'L','O','G','I','T','E','C','H',' ','I','N','C','.','
',' ','1','0','/','1','0','/','9','7','\r'};
```

Als erstes wird die serielle Schnittstelle, es handelt sich dabei um die Schnittstelle an der das Nullmodemkabel angeschlossen ist, vorbereitet. Im nachstehenden Programmteil wird ein Filedeskriptor auf die serielle Schnittstelle gesetzt, die alten Einstellungen der Schnittstelle werden gespeichert, und mittels diverser Flags werden neue Einstellungen für die Schnittstelle vorgenommen. Danach ist das Interface zur Kommunikation bereit.

```
/*Open modem device for reading and writing and not as controlling   tty
because we don't want to get killed if linenoise sends CTRL-C.*/
```

```
        fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
        if (fd < 0 ) {perror(MODEMDEVICE); exit(-1); }
/* save current serial port settings */
        tcgetattr(fd,&oldtio);
/* clear struct for new port settings */
        bzero(&newtio, sizeof(newtio));

/*BAUDRATE: Set bps rate. You could also use cfsetispeed and   cfsetospeed.
  CRTSCTS : output hardware flow control (only used if the cable has
                  all necessary lines. See sect. 7 of Serial-HOWTO)
  CS8     : 8n1 (8bit,no parity,1 stopbit)
  CLOCAL  : local connection, no modem contol
  CREAD   : enable receiving characters*/
        newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD |
CSTOPB;

/*IGNPAR  : ignore bytes with parity errors
  ICRNL   : map CR to NL (otherwise a CR input on the other computer
            will not terminate input)otherwise make device raw
           (no other input processing)*/
        newtio.c_iflag = IGNPAR | ICRNL;

/*Raw output*/
        newtio.c_oflag = 0;

/*ICANON  : enable canonical input disable all echo functionality, and
            don't send signals to calling program*/
        newtio.c_lflag = ICANON;

/*initialize all control characters default values can be found in
/usr/include/termios.h, and are given in the comments, but we don't need
them here*/
        newtio.c_cc[VINTR]    = 0;     /* Ctrl-c */
        newtio.c_cc[VQUIT]    = 0;     /* Ctrl-\ */
        newtio.c_cc[VERASE]   = 0;     /* del */
        newtio.c_cc[VKILL]    = 0;     /* @ */
        newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
        newtio.c_cc[VTIME]    = 0;     /* inter-character timer unused */
        newtio.c_cc[VMIN]     = 1;     /* Session ing read until 1
character
                                          arrives */
        newtio.c_cc[VSWTC]    = 0;     /* '\0' */
        newtio.c_cc[VSTART]   = 0;     /* Ctrl-q */
        newtio.c_cc[VSTOP]    = 0;     /* Ctrl-s */
        newtio.c_cc[VSUSP]    = 0;     /* Ctrl-z */
        newtio.c_cc[VEOL]     = 0;     /* '\0' */
        newtio.c_cc[VREPRINT] = 0;     /* Ctrl-r */
        newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
        newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
        newtio.c_cc[VLNEXT]   = 0;     /* Ctrl-v */
        newtio.c_cc[VEOL2]    = 0;     /* '\0' */

/*now clean the modem line and activate the settings for the port*/
        tcflush(fd, TCIFLUSH);
        tcsetattr(fd,TCSANOW,&newtio);

/*terminal settings done, now handle input In this example, inputting a 'z'
at the beginning of a line will exit the program.*/
```

In einer Endlosschleife wird nun die serielle Schnittstelle abgehört und auf die Anfragen des SpaceMouse-Treiber entsprechend reagiert.

```
/*forever*/
while(1)
      {
/*reading data from seriell port*/
      res = read(fd,buf,sizeof(buf));

/*identifing the request and give prober response*/
      switch(buf[0])
            {
            case 'v' :
                  {
                  send = write(fd,version,sizeof(version));
                  printf("version send\n");
                  } break;
            case 'm' :
                  {
                  if(buf[1] == 'Q')
                  send = write(fd,status_mode,sizeof(status_mode));
                  send = write(fd,buf,sizeof(buf));
                  printf("status_mode send\n");
                  } break;
            case 'c' :
                  {
                  if(buf[1] == 'Q')
                  send = write(fd,compress_mode,sizeof(compress_mode));
                  send = write(fd,buf,sizeof(buf));
                  printf("compress_mode send\n");
                  } break;
            case 'l' :
                  {
                  if(buf[1] == 'Q')
                  send =write(fd,led_mode,sizeof(led_mode));
                  send = write(fd,buf,sizeof(buf));
                  printf("led_mod send\n");
                  } break;
            case 'q' :
                  {
                  if(buf[1] == 'Q')
                  send = write(fd,sensetiv_mode,sizeof(sensetiv_mode));
                  send = write(fd,buf,sizeof(buf));
                  printf("sensetiv_mod send\n");
                  } break;
            case 'p' :
                  {
                  if(buf[1] == 'Q')
                  send = write(fd,datarate_mode,sizeof(datarate_mode));
                  send = write(fd,buf,sizeof(buf));
                  printf("datarat_mod send\n");
                  } break;
            case 'n' :
                  {
                  if(buf[1] == 'Q')
                  send =write(fd,nullradius_mode,sizeof(nullradius_mode));
                  send = write(fd,buf,sizeof(buf));
                  printf("nullradius_mode send\n");
                  } break;
            case 'b' :
```

```
                {
                send = write(fd,beep,sizeof(beep));
                printf("beep send\n");
                beepcount++;
                printf("beepcount:%i\n",beepcount);
                } break;
           }//end switch

     }//end while
```

Abschließend werden die ursprünglichen Einstellungen der seriellen Schnittstelle wieder hergestellt und der Filedeskriptor geschlossen.

Das Dämonprogramm muss vor dem SpaceMouse-Treiber gestartet werden, um gleich auf die ersten Anfragen des Treibers reagieren zu können. Außerdem muss es während der Verwendung des Gerätes im Hintergrund weiterlaufen, da manche Anwendungen per Treiber neue Konfigurationen an das Eingabegerät schicken, da sie glauben sie wären mit einer SpaceMouse verbunden.

Das Hauptprogramm kann gestartet werden nachdem das Dämonprogramm sich mit dem Treiber synchronisiert hat. Auch hier werden natürlich erst wieder Bibliotheken geladen, Makros, Variablen und Konstanten vereinbart. Danach werden hier beide seriellen Schnittstellen zur Kommunikation bereit gemacht. An der einen Schnittstelle ist das Nullmodemkabel für die Kommunikation mit dem Zielrechner angeschlossen. An der anderen Schnittstelle ist unser Eingabegerät angeschlossen (in diesem Fall der kleine Kugelfisch), um den Input abzufragen. Der Quellcode für die eben angeführten Schritte ähnelt dem des Dämonprogramms sehr, deswegen soll darauf an dieser Stelle verzichtet werden.

Weiterhin werden auch für die USB-Schnittstellen Filedeskriptoren angelegt um von ihnen lesen zu können.

```
/*open USB-ports for reading inputdata and give errormessgage if it’s not
possible*/
  if ((fd1 = open("/dev/input/event1", O_RDONLY)) < 0)
      {
      perror("error: can't open 1. USB-Device");
      exit(1);
      }

  if ((fd2 = open("/dev/input/event2", O_RDONLY)) < 0)
      {
      perror("error: can't open 2. USB-Device");
      exit(1);
      }
```

Anschließend werden in einer Endlosschleife die Inputdaten eingelesen.

```
/*forever*/
while (1)
    {
        /*reading data from serial-port (inputdevice) and give errormessage
        if it dosn´t work*/
        re = read(fdread,readbuf,sizeof(readbuf));

        if (re == -1)
            {
            perror("error while reading from serialport 0");
            }

/*reading data from 1. and 2. USB-port (inputdevice) and give
      errormessage if it dosn´t work*/
      read_bytes_ev1 = read(fd1, &event_1_buffer, sizeof(struct
                            input_event));
      if (read_bytes_ev1 < (int) sizeof(struct input_event))
            {
            perror("evtest: short read");
            exit (1);
            }

      read_bytes_ev2 = read(fd2, &event_2_buffer, sizeof(struct
                            input_event));
      if (read_bytes_ev2 < (int) sizeof(struct input_event))
            {
            perror("evtest: short read");
            exit (1);
            }
```

Immer noch in der Endlosschleife werden die Daten bearbeitet und auf das SpaceMouse-Protokoll angepasst. Dies geschieht auf unterschiedliche Weise für die beiden Manipulationsarten. Die Translationswerte liegen schon im richtigen Format vor, da sie ja von der verbauten SpaceMouse-Sensorik kommen. Sie müssen nur umsortiert, also an eine andere Stelle in der Zeichenkette verschoben werden, damit sie der Treiber richtig interpretiert. Aufgrund der Art und Weise wie die SpaceMouse-Sensorik eingebaut wurde, sind aber leider nicht nur die Achsen vertauscht, sondern bei zwei Achsen auch die Richtungen (+,-). Dies hat zur Folge, dass die Inputdaten nicht nur an die richtige Stelle in der Zeichenkette verschoben, sondern zuvor noch in ihr Komplement überführt werden müssen. Der nachfolgende Quellcode zeigt dies für die Y-Achse.

```
//correcting wrong Y-translation-data
       i = 0;
       for( count = 1; count < 5; count++)
            {
            for (index = 0; index < 16; index++)
                  {
                  if(readbuf[count] == nibble_table[index])
                        {
                        transy[i] = index;
                        i++;
                        }
```

```
                  }
            }

      if( werty < 0)
            {
                  werty = abs(werty);
                  buf[5] = 'H';
                  bufcount=8;
                  while(werty !=0 )
                        {
                        writet = werty%16;
                        buf[bufcount]=nibble_table[writet];
                        werty = werty/16;
                        bufcount--;
                        }
            }
      else if ( werty > 0 )
            {
                  werty = werty * -1;
                  buf[5] = 'G';
                  werty=4093+werty;
                  bufcount=8;
                  while(werty !=0 )
                        {
                        writet = werty%16;
                        buf[bufcount]=nibble_table[writet];
                        werty = werty/16;
                        bufcount--;
                        }
            }
```

Für die Rotationswerte ist es erforderlich, sie erst auf das SpaceMouse-Protokoll anzupassen. Dies wird wie folgt gemacht:

In der Praxis zeigte sich, dass sehr selten Inputwerte größer 30 bzw. kleiner -30 auftreten, wenn man normal an der Kugel dreht. Daher werden nur Werte in diesem Intervall (-30 bis 30) weiterverarbeitet. Diese Werte werden mit 68 multipliziert, so dass ein Intervall von (-2040 bis 2040) resultiert. Dieser Wertebereich ist zwar nur ein Bruchteil des theoretischen Wertebereichs, hat sich aber als ausreichend erwiesen. Nachfolgend die Implementation dazu:

```
switch (event_1_buffer.code)
            {
            case 0 :{
                    if ((event_1_buffer.value < 0) && (event_1_buffer.value
                                                      > -30))
                        {
                        buf[17] = 'G';
                        gdiv=4093-((abs(event_1_buffer.value))*68);
                        bufcount=20;
                        while(gdiv !=0 )
                              {
                              writet = gdiv%16;
                              buf[bufcount]=nibble_table[writet];
                              gdiv = gdiv/16;
```

```
                              bufcount--;
                              }
                        }
                  else if ((event_1_buffer.value > 0) &&
                           (event_1_buffer.value < 30))
                        {
                        buf[17] = 'H';
                        gdiv=(event_1_buffer.value)*68;
                        bufcount=20;
                        while(gdiv !=0 )
                              {
                              writet = gdiv%16;
                              buf[bufcount]=nibble_table[writet];
                              gdiv = gdiv/16;
                              bufcount--;
                              }
                        }
                  break;
                  }
```

Die entsprechenden Operationen werden für alle sechs Freiheitsgrade ausgeführt. Als letzte Aktion der Schleife wird das entsprechend zusammengestellte Datenpaket an den Zielrechner geschickt.

```
    //writing on serial-port
            wr = write(fdwrite,buf,sizeof(buf));

            if (wr == -1){perror("error while writing on serialport 1");}
} // end while
```

Abschließend werden die einzelnen Filedeskriptoren wie üblich geschlossen.

```
  //close all file discriptors
  close(fd1);
  close(fd2);
  close(fdwrite);
  exit(0);
}//end main
```

Für Unix-Betriebssystemen ist anzumerken, dass die USB-Events zeitlich vergeben werden. D.h. die Reihenfolge des Einsteckens der USB-Anschlüsse spiegelt sich in der laufenden Nummer der Events wieder. Es ist also wahrscheinlich, dass das Programm eine Fehlermeldung produziert, da die Filedeskriptoren mit festen Pfadangaben initialisiert werden und diese nun neu vergeben worden sind.

Des weiteren sind die Bezeichnungen für die seriellen Schnittstellen fest vergeben. Es muss darauf geachtet werden, dass die SpaceMouse-Sensorik an ttyS0 (meist der obere der beiden Anschlüsse am Board) und das Nullmodemkabel an ttyS1 (logischerweise meist der untere Anschluss) angesteckt wird.

8. Zusammenfassung

Die Performance von vier 6DoF Eingabegeräten wurde anhand einer einfachen Docking Aufgabe bewertet. Drei dieser Geräte wurden an der BUW entwickelt und basieren auf einem neuen Interaktionskonzept. Rotationen werden dabei durch Drehen einer Kugel gemessen. Die Kugel dient dabei als Verbindung/Prob zwischen der realen Welt und einer virtuellen Szene. Die Position bzw. die Orientierung der Kugel wurde direkt auf das virtuelle Objekt übertragen. Die Kugel wird z.B. in eine bestimmte Richtung um einen bestimmten Betrag gedreht. Das Szenenobjekt rotiert genau entlang der selben Richtung um diesen Betrag bzw. um einen faktorisierten Betrag. Visuelles und motorisches Feedback gehen also miteinander einher. Diese Form der Rotationssteuerung sollte herkömmlichen Techniken überlegen sein. Bei der SpaceMouse wird Rotation durch Kippen des Pucks ausgelöst. Die Inputbewegungen sind asynchron zur ausgelösten Cursorbewegung. Die SpaceMouse mit ihrer gänzlich anderen, kommerziell jedoch bewährten, Inputtechnik, diente als unser Referenzgerät.

Im Test schnitten unsere kugelgesteuerten Geräte deutlich besser ab als die SpaceMouse. Und zwar sowohl bei den TCTs als auch bei den subjektiven Aussagen der Versuchspersonen. Unser Konzept, dass positionsgesteuerte Rotation einer Kugel geschwindigkeitsgesteuerter Rotation überliegt, konnte bestätigt werden.

Diese Erkenntnis wird Ausgangspunkt für die Entwicklung weiterer Kugelgesteuerter Eingabegeräte an der BUW sein. Besonders die konstruktionstechnische Umsetzung stellt noch eine große Herausforderung dar. Das Versagen des großen Kugelfischs zeigt deutlich, dass die Entwicklung geeigneter Messtechniken/-sensoriken viel Know How benötigt. Die Verwendung von ausgereifter SpaceMouse-Messtechnik zur Translationserfassung und Trackballsensoren zur Erfassung der Rotationsinputs bei Kugelmaus und kleinem Kugelfisch, hat sicherlich zum Erfolg dieser Geräte beigetragen.

Interessanterweise haben unterschiedlich komplexe Interaktionstrategien zu ähnlicher Performance geführt. So war die Kugelmaus mit strikter Trennung von Translation und Rotation genauso effizient wie der kleine Kugelfisch, wo Translation und Rotation simultan ausführbar waren/benutzt worden sind. Einzelaktionen sind

immer noch genauer als simultane Vorgänge. Der Zeitgewinn durch Simultanität wird durch niedrigere Genauigkeit ausgeglichen. Die Erhöhung der Genauigkeit bzw. die Eingabeleichtigkeit von simultanen Aktionen wird sicherlich eine der Herausforderung bei der Entwicklung zukünftiger Eingabegeräte sein.

9. Quellen und Literatur

Boritz J. and Booth K. S., „A Study of Interactive 6 DOF Docking in a Computerized Virtual Environment", Department of Computer Science at the University of British Columbia Vancouver

Buxton, W. „An Introduction to Human Input to Computers", (2003)

Europäische Hochschulschriften „Interventionsprogrammes", Frankfurt am Main 2002

Göbel M., and Friesdorf W., "Evaluation of Input devices for 3D-navigation in medical applications", Institute of Work Science and Product Ergonomics, Berlin University of Technology

Jacob, R. J. K., Sibert, L. E., McFarlane, D. C., & Mullen, M. P. (1994). Integrality and separability of input devices. *ACM Transactions on Computer-Human Interaction, 1*(1),
3-26.

Jordan, W. Patrick: „An introduction to usability", Taylor & Francis, London 2002

Masliah, M. R., „Pilot Experiment on Measuring Coordination"

Masliah, M. R., and Milgram, P. „Measuring the Allocation of Control in a 6 Degree-of-Freedom Docking Experiment." *CHI 2000 Conference on Human Factors in Computing Systems*, The Hague, Netherlands.

Priemuth, Kathrin: „Motivation und Volition: Evaluation eines psychologischen

Prof. C. Wüthrich, Vorlesung „Animationssysteme", BUW

Tomas Akenine-Möller „Real-Time Rendering", Hauptbibliothek, Lehrbuchsammlung, Sgn.: 194 774

Wickens, C. D. (1998), „An Introduction to Human Factors Engineering"

Wottawa, Heinrich und Thierau, Heike: "Lehrbuch Evaluation", 2. Auflage, Huber Verlag, Bern 1998

Zhai, S. (1998), "User Performance in Relation to 3D Input Device Design" Computer Graphics 32(4) pp50-54 © ACM

Zhai, S., Buxton, W., & Milgram, P. (1994). The „silk cursor": investigating transparency for 3D target acquisition. In *Proc. of CHI'94: ACM conference on Human Factors in Computing Systems*, (pp. 459-464). Boston

Zhai, S. (1995). "Human Performance in Six Degree of Freedom Input Control," Ph.D., University of Toronto, Toronto.

Zhai, S., and Milgram, P. “Quantifying Coordination in Multiple DOF Movement and Its Application to Evaluating 6 DOF Input Devices.” *Proceedings of the Conference on Human Factors in Computing Systems CHI '98*, Los Angeles, 320-327.

Zhai, S., and Senders, J. W. “Investigating coordination in multidegree of freedom control I: time-on-target analysis of 6 DOF tracking.” *41st Annual Meeting of Human Factors and Ergonomics Society*, Albuquerque, 1249-1253 und 1254-1258

Internetquellen

http://www.megatron.de/Wegsensoren/Ubersicht_potentiom_/Shortview_MM/shortview_mm.html, letzter Aufruf 01. 11. 2004

http://www.vrib.de (Abb. 15), letzter Aufruf 01. 11. 2004

http://www.hama.com , letzter Aufruf 01. 11. 2004

Non-Isomorphic 3D Rotational Techniques“ von Ivan Poupyrev, Suzanne Weghorst, Sidney Fels

3D Motion Control Serial Protocol, http://www.3dconnexions.com , letzter Aufruf 01. 11. 2004

http://www.isner.com/tutorials/quatSpells/quaternion_spells_14.htm#visualizing%20Rotations, letzter Aufruf 01. 11. 2004

http://we.home.agilent.com/USeng/nav/-536893734.536883756/pd.html, letzter Aufruf 01. 11. 2004

http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-1.html#node_toc_node_chap_Temp_1, letzter Aufruf 01. 11. 2004

http://www.sbg.ac.at/evaluation/frageb_konzept.html, letzter Aufruf 01. 11. 2004

http://www.evaluieren.de/infos/links/index.htm , letzter Aufruf 01. 11. 2004

http://www.psitronic.de/t...00310000000000000000 , letzter Aufruf 01. 11. 2004

http://www.techinfo.rwth-...ntl-db.pl?marrenbach , letzter Aufruf 01. 11. 2004

Folgende Kapitel wurden von den jeweiligen Projektteilnehmern bearbeitet:

Abstract	Verena Skuk
1. Einführung 1.1. EvaGrip2 1.2. Begriffe und Grundlagen 1,2.1. Gerätetypen 1.2.2. Steuerungsmodi	Manuel Gleisberg
1.2.3. Zhai's DockingTask	Carsten Tetens
2. Eingabegeräte 2.1. Referenzgerät, SM 2.2. Motivation für neue Geräte 2.3. grKF 2.4. klKF 2.5 KM 2.6. Geräteüberblick 3. Hypothesen	Robert Gerling
4. Versuchsmodalitäten 4.1.Versuchsdesign 4.2. Versuchspersonen 4.3. Ablauf 4.4. Versuchsaufbau 4.5. Testaufgabe 4.6. Fragebogendesign 4.7. Ablaufplan	Carsten Tetens
5. Implementation 5.1. Einleitung 5.1.1. Avango 5.1.2. Scheme 5.2. grober Ablauf 5.3. Feinheiten der Programmierung. 5.3.1. config.txt	Verena Skuk
5.3.2. Geräteimplementierung	Carsten Tetens
5.3.3. Szenerie 5.3.4. Translation und Rotation 5.3.5. Cursorstartpositionen 5.3.6. ScaleDCD 5.3.7. Toleranz und Distanzvektoren	Verena Skuk
5.3.8. Writeout 5.3.9. Callback	Manuel Gleisberg
5.3.10. Verzeichnisstruktur	Verena Skuk
5.4. Quaternionen 5.4.1. Grundlage Mapping 5.4.2. Quaternionen 5.4.3. Geometrische Erklärung 5.4.4. Quaternionenimplementation	Andre Kunert

6. Versuchsauswertung	
6.1. TCT	Manuel Gleisberg
6.2. Simultanitäten	Andre Kunert, Verena Skuk
6.3. Fragebögen	Robert Gerling
7. Konverter	Robert Gerling
8. Zusammenfassung	Andre Kunert